Head Office: Università degli Studi di Padova

Department: General Psychology

Ph.D. Course: Brain, Mind, and Computer Science

Curriculum: Computer Science and Innovation for Societal Challenges

Series: XXXVII (2021-2024)

Algorithmic reasoning in large language models and neuro-symbolic architectures

Coordinator: Prof. Anna Spagnolli

Supervisor: Prof. Alessandro Sperduti

Co-Supervisor: Prof. Alberto Testolin

Ph.D. student: Flavio Petruzzellis

Algorithmic reasoning in large language models and neuro-symbolic architectures

by

Flavio Petruzzellis

Submitted to the Department of General Psycology
on September 30, 2024 in partial fulfillment of the requirements for the degree of

BRAIN, MIND & COMPUTER SCIENCE

ABSTRACT

Rule-based systems from the tradition of artificial intelligence possess reliable reasoning capabilities, but struggle with flexibility and scalability. Conversely, modern deep learning-based systems have demonstrated impressive performance in a wide variety of tasks requiring "fast thinking", but lack robust reasoning capabilities. Motivated by the need to overcome these limitations, this thesis investigates the design of AI systems that can learn to reason systematically. Drawing from a rich tradition of AI and cognitive science, we leverage the concept of compositionality as a common thread in our research. The contributions we offer are twofold. First, we investigate the systematic reasoning capabilities of Large Language Models probing them on the simplification of nested mathematical formulas, a problem that reflects key properties of compositionality. Our experimental results on Llama 2 and GPT models reveal that while scaling LLMs brings some improvements, their systematic reasoning capabilities remain limited, even with specialized prompting methods like chain-of-thought reasoning. Second, we propose a neuro-symbolic framework designed to learn and execute convergent term rewriting systems, which can formally describe simple algorithms for the iterative simplification of nested mathematical formulas. The framework is implemented in two variants: NRS and FastNRS. We benchmark these implementations against the Neural Data Router, OpenAI's GPT-4 and o1-preview, demonstrating the robust systematic reasoning capability of our neuro-symbolic approach, while critically assessing the impact of our design choices on efficiency and their limitations.

Thesis supervisor: Alessandro Sperduti
Title: Professor of Computer Science

Thesis supervisor: Alberto Testolin
Title: Professor of Computer Science

Acknowledgments

A Ph.D. when you're 24 years old is a long journey. This thesis would have never been written without the invaluable support of many people I had the good fortune of meeting in my life before and during this journey.

To Alberto Testolin, thank you for sparking and igniting interest in cognitive science and neural networks in me for the first time. Your teachings allowed me to explore fields of knowledge that would have otherwise remained unknown. To Alessandro Sperduti, thank you for embarking in this journey with me and helping me trace the path. Your experience and dedication have been fundamental to find the light and not getting lost. To both of you, thank you for nurturing critical thinking in me towards the goal of making good science.

To Fabiola, who was beside me long before and will be there long after. There are many things I am grateful for. Thank you for not being the ordinary girlfriend. Thank you for walking this journey with me and enduring the hard part of it day by day. Thank you for truly caring for me and always making me see things from a different angle. Thank you for pushing me to grow a better person in this journey. Without you, it would not have been the same.

To my family: my sister Sveva, my mother Antonella, my father Silverio and my grandfather Salvatore, who were there for me during this journey, no matter where we were. Your support in my whole life has been much greater than I ever acknowledged, and I would have never been living the life I live today without it. For this, I deeply thank you.

To Vincenzo and Sabina, who saw the beginning of this journey, my first steps in it from close by and the next ones from the distance. Thank you for being there for me and being my friends. Sharing parts of this journey with you really made a difference for me.

To Alessandro Padella, thank you for being there for me when I needed it the most and helping me think through the hard decision. You are a good friend and I was lucky to meet you.

To my colleagues in the Brain, Mind & Computer Science program. Thank you for the support we always gave to each other. Sharing this journey with you, which would have otherwise been so much on each of our shoulders, made it lighter and more pleasant.

To Corrado Monti, thank you for being there when I needed advice and for hosting me to present the first fruits of my research work. You made me feel connected to a network in times when this was hard to see.

To Pietro Liò, thank you for having me in Cambridge and giving me the opportunity to meet people whose names I had just read on a screen.

To Francesco Caso, Lorenzo Petrosino, Simone Monaco, Samuele Fonio, and all the people I spent great times with in the UK. Thank you for making it fun.

To Alberto Fernandez Hilario, Luzia Straßer and the team of the Arqus mentoring programme for early-stage researchers: thank you for your amazing work. Having the opportunity to take part in the programme made me feel heard. To the European Union: thank you for putting up strategies that aim to tackle systemic challenges in a world where we mostly feel them like individual problems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis offers a contribution to the debate on a central and current topic in machine learning and artificial intelligence: learning systematic reasoning procedures from data. Creating artificial systems that possess general-purpose reasoning capabilities is a paramount goal in artificial intelligence, and it has persisted in the history of the discipline. In this chapter, we consider the broader context in which our investigations are placed. We start with an overview on the two dominant paradigms in AI systems: the one based on rules and formal reasoning that was predominant in AI systems developed until the 1990s, and the machine learning-based one that has become prevalent afterwards. By considering the limitations of each paradigm, we motivate the need of research on *learning* reasoning procedures from data, and give an overview of different perspectives that are currently competing in the scientific debate on this topic. Finally, we describe the contributions offered in this thesis and the research questions that guide our investigation.

1.1 Reasoning and learning in AI

Artificial intelligence (AI) has long been driven by the ambition to replicate and even surpass human cognitive abilities, with abstract reasoning representing the paramount goal of this scientific endeavor. Abstract reasoning—the ability to manipulate concepts, solve problems and plan in the long term—has historically been considered a uniquely human capability, setting us apart from other animals. This focus on reasoning has shaped the early development of AI, where the primary goal was to create systems capable of solving formal reasoning problems, mirroring human intellectual tasks.

In its early days, AI research concentrated heavily on the logical aspects of intelligence. Researchers sought to create algorithms that could replicate the structured and rule-based processes humans use to solve complex problems, especially in highly specialized domains and scientific disciplines.

One of the earliest and most well-known examples of AI systems designed with formal reasoning capabilities is the Logic Theorist, developed by Allen Newell and Herbert A. Simon in the mid-1950s [1]. The Logic Theorist was designed to prove mathematical theorems, a task that required sophisticated deductive reasoning. The program successfully proved 38 of the first 52 theorems in Whitehead and Russell's Principia Mathematica, including a proof

that was more elegant than the original. This achievement demonstrated the potential of AI to tackle complex reasoning tasks and laid the groundwork for subsequent developments in the field.

Another seminal AI system designed to implement formal reasoning procedures was the chess-playing program developed by Claude Shannon and later improved by others [2]. Chess, a complex game requiring the finest strategic skills developed through years of study and practice, was seen as the perfect testbed for AI's reasoning capabilities. The success of these programs, culminating in IBM's Deep Blue defeating world chess champion Garry Kasparov in 1997, was a landmark achievement. It demonstrated that AI could indeed perform at superhuman levels in domains requiring rigorous logical reasoning.

These early AI systems, grounded in formal logic and symbolic reasoning, set the stage for the evolution of the field. However, the limitations of these approaches also became apparent, as these systems and their successors could only be applied to the specific domains they were programmed for, requiring significant human labor to be adopted and maintained, leading to the exploration of alternative learning-based methods, including the deep learning approaches that dominate AI research today.

Starting from the 1990s, the focus of AI research has shifted markedly from the early emphasis on formal reasoning to the learning aspect of intelligence. This shift has been heavily influenced by advances in neuroscience and cognitive science, where the understanding of how the brain learns and processes information has provided inspiration for the development of artificial neural networks and deep learning systems. One notable example is the structure of convolutional neural networks: CNNs are designed with multiple layers that progressively capture more abstract features of an image [3], just as the visual cortex processes visual information through a hierarchy of layers. Another significant parallel between deep learning and neuroscience is found in the backpropagation algorithm [4] used to optimize artificial neural networks, which resembles the adjustment of connection strengths between biological neurons based on feedback, following the principles of Hebbian learning [5, 6].

Deep learning models have become the cornerstone of modern AI, driving breakthroughs in various applications. These systems are exceptionally good at identifying patterns in vast datasets, making them useful for processing natural data in different modalities, including text [7], images [8, 9], speech [10], also across modalities [11, 12] and for a wide variety of tasks [13]. Generally speaking, deep learning models excel at tasks requiring intuitive, fast, and automatic thinking—often called System 1 tasks, following Daniel Kahneman's dual-process theory of cognition [14]. Moreover, large language models (LLMs), which have recently emerged as the most advanced application of this approach in AI, seem to push the boundaries of what can be achieved with neural technology. LLMs can convincingly simulate reasoning in specific contexts, such as solving school-level problems in various domains. In some scenarios, they can even pass the Turing Test, fooling humans into believing they are conversing with another human. While still a benchmark for evaluating AI, the Turing Test is increasingly seen as insufficient to measure true reasoning capabilities, as people can frequently fail to distinguish other people from LLMs in real-world scenarios [15, 16].

Despite these impressive achievements, the capacity of current AI systems to perform tasks requiring System 2 thinking in Kahneman's theory [14] —deliberative, slow, and logical reasoning—remains a significant challenge. Large language models fail to understand and apply abstract reasoning in novel situations genuinely [17], and can generate plausible but

fundamentally shallow or incorrect answers when faced with complex reasoning tasks [18].

Thus, as the impressive capabilities of deep learning systems reach their peak and start to show their limitations, a general consensus is forming on the need for novel approaches to overcome the limitations of both the formal reasoning-based paradigm and the learning-based paradigm while preserving their strengths.

1.2 Towards machines that can learn to reason

Human beings, from a relatively early age, develop the remarkable ability to engage in abstract thinking, a cognitive capability that plays a critical role in navigating complex environments. Abstract thinking can be understood temporally, as thoughts that relate to long-term goals or to events in the past or future: for example, if someone wanted to reach a goal in the future (e.g., 'buy a car'), they could think of steps to accomplish this goal or to past actions that led the person to set this goal (e.g., 'getting a driving licence'). Alternatively, it can be understood relationally, as thoughts that emphasize the relationships between abstract representations of real-world entities rather than merely responding to immediate stimuli, like humans who were trained in mathematics can do to solve problems using numbers and mathematical operations [19, 20]. Some aspects of abstract thinking are visible, at least superficially, in contemporary AI systems, such as large language models exhibiting sophisticated linguistic abilities and simulating aspects of reasoning, or advanced reinforcement learning (RL) systems discovering complex, long-term strategies [21]. However, a central component of abstract thinking is the use of symbols, which allows humans to learn and manipulate representations of both real-world and imaginary entities to solve both practical and highly formalized problems. This capability is arguably still out of reach for current AI systems: indeed, how we could build artificial systems capable of learning to interact with formal symbolic representations systematically, i.e., with a high degree of regularity and predictability, remains an open question.

While deep learning has been considered the most promising AI technology in recent years, whether connectionist systems can process symbolic structure is a long-standing question in cognitive science and AI, rooted in the debate over compositionality [22]. Some linguists and cognitive scientists have argued that, because language exhibits compositional properties, and it can be understood by thought, then thought must also be compositional [23]. From this perspective, achieving human-like symbol processing in AI requires systems capable of handling compositional structures and, consequently, capable of compositional reasoning. Nowadays, several competing views are debated among researchers and scholars on how to create systems that can reliably learn to manipulate symbolic systems and apply systematic reasoning procedures.

An approach proposed to achieve this goal focuses on the role of symbols and the emergence of meaning through interactions [24]. Since symbols are a fundamental aspect of abstract thinking, this line of research posits that the development of intelligent systems must involve the ability to learn, manipulate, and even invent symbolic representations, allowing for a richer and more meaningful engagement with both real-world and conceptual problems [25].

Proponents of the autoregressive foundational models and large language models (LLMs)

argue that these architectures offer a promising path to human-like and even super-human reasoning [26, 27]. They claim that LLMs exhibit surprising capabilities which emerge as a consequence of training on vast and diverse datasets. As these models are scaled up and further refined, supporters argue, their ability to simulate human-like reasoning will grow more robust. In particular, they point to emergent behaviors in LLMs, such as few-shot learning and the ability to generate coherent and contextually appropriate responses in novel situations, as evidence that these models could eventually reach System 2-like thinking.

Other researchers are skeptical regarding the possibilities of current autoregressive models and advocate for a paradigm shift to bridge the gap between current AI capabilities and System 2 reasoning [28, 29]. This line of research is less concerned with directly building reasoning capabilities in AI systems but is instead more focused on creating agents that can perform tasks that, to this date, animals can accomplish but machines cannot. The approach proposes to do so by constructing world models and hierarchical representations, drawing inspiration from neuroscience and the natural world [30]. This would lead to the creation of artificial systems that have animal- and human-like learning biases that could be the basis for the development of flexible reasoning in the real world.

Finally, a third approach seeks to integrate symbolic methods from classical AI with the strengths of modern neural systems [31–33]. This hybrid model aims to combine the robust, rule-based reasoning typical of symbolic AI with the flexibility and learning capabilities of neural networks. Proponents of the neuro-symbolic approach argue that this integration is necessary because purely neural and symbolic models have opposite and complementary strengths and weaknesses. While classical AI and symbolic methods based on logic and reasoning can provide strong generalization guarantees, neural networks enable adaptation and pattern recognition. Therefore, building systems that integrate the two approaches in various ways could be a viable pathway toward achieving a form of human-like reasoning.

1.3 Statement of contributions

This thesis is informed by the interdisciplinary literature in cognitive science and AI on compositional reasoning and systematic generalization. Drawing from this rich background and considering the research directions described above, we offer several contributions which are organized in the thesis as follows.

Chapter 2 provides a broad overview of the concept of compositionality, a foundational idea in cognitive science and AI that refers to the way complex structures can be built from simpler components. Throughout the thesis, we will consider reasoning problems that can be solved using highly systematic and reliable symbol manipulation procedures, such as simplifying nested mathematical formulas. The concept of compositionality is key to describe both the properties of the symbolic inputs that will be under study and the information processing systems that manipulate them.

The chapter begins with a historical background, tracing the origins of the debate on compositionality in connectionist architectures. Following this, the discussion shifts to recent research on compositionality within modern neural architectures, focusing on novel theoretical frameworks, benchmarks and models, and providing relevant technical background for our contributions. We then consider the related research area of learning algorithms with

neural networks, giving a detailed account of recent efforts in this direction. Finally, the chapter considers the technical developments that lead to current large-language models and examines the available evidence on their compositional and algorithmic reasoning abilities.

As we have mentioned in Section 1.1, large language models are regarded as the most successful and widely deployed AI technology developed to date. They have gained attention for their language manipulation capabilities, which have shown remarkable effectiveness in a wide range of tasks in various domains. Furthermore, as we described in Section 1.2, some researchers consider them to be the most promising paradigm for learning robust systematic reasoning procedures. In this area of research, it has been argued that such skills emerge from the proper combination of prompting methods and the scale of the models themselves [34]. To critically evaluate these claims, this thesis addresses three key research questions:

1. Does the capability to reason systematically emerge as the model scale increases?
2. Can this capability be reliably elicited through careful prompting?
3. Overall, can we say that LLMs genuinely learn to reason systematically?

To answer these questions, in Chapter 3, we present a series of experiments that benchmark various large language models from the GPT and Llama families on the simplification of mathematical formulas, a type of task that requires reliable systematic reasoning, as we mentioned earlier. The results of this analysis provide novel evidence on whether large language models develop reasoning capabilities and of what kind these might be. Specifically, we find that both scaling and fine-tuning models in the Llama 2 family improve their performance only on relatively simple problems and do not make reliable systematic reasoning capabilities emerge in these models. Similarly, we find that specialized prompting methods designed to improve reasoning capabilities, while improving overall performance of GPT models on the tasks we consider, cannot elicit in these models reasoning capabilities with algorithmic-like generalization properties. We also consider the Neural Data Router, which we consider as a representative of small-scale neural models specialized to solve algorithmic reasoning tasks, aiming to understand the limitations of these architectures.

Additionally, this thesis offers a contribution to research efforts focused on the development of novel artificial intelligence systems that can learn to reason systematically. The contribution in this direction closely aligns with two research areas: on the one hand, it is related to the research direction proposing to integrate symbolic and neural information processing to leverage the strengths of both paradigms. On the other hand, our proposal can be inserted in the stream of research that aims to develop learning-based AI systems that exhibit systematic generalization similar to those possessed by classical algorithms, where consistent and reliable performance is observed not just on problems seen during training but also on novel, more complex problems that possess specific structural characteristics (for an overview of this research area, we address the reader to Section 2.3).

In Chapter 4, we present a novel framework to learn algorithms in the class of convergent term rewriting systems with a hybrid neuro-symbolic architecture. The problems this class of algorithms can solve include the ones considered in the previous chapter. In this chapter, we thus describe the class of problems more formally and explore several critical research questions that guided the development of a specialized hybrid neuro-symbolic system:

1. How can a hybrid neuro-symbolic architecture be designed specifically for the class of problems we consider?

2. What is the impact of different design elements on the system's performance?

3. What is the impact of these design choices on efficiency, measured in terms of training time, inference time, and memory consumption?

To answer the first question, we thoroughly analyze the class of algorithms implemented by convergent term rewriting systems and propose a formalization of the problems that can be solved with those. Based on this formalization, we derive the proposed neuro-symbolic framework which serves as an architectural blueprint that can be potentially implemented in neuro-symbolic systems in different ways. Two specific implementations of this framework are then described in detail: the Neural Rewriting System (NRS) and the Fast Neural Rewriting System (FastNRS), which differ in design choices made on key modules of the architecture. Importantly, we demonstrate that both systems can be trained in a multi-domain setting where the same architecture learns to solve problems in different domains without requiring ad-hoc architectural modifications, thereby effectively learning multiple convergent term rewriting systems at once.

We present empirical evidence on the performance of both systems on four datasets of nested mathematical formulas from diverse domains. We compare the performance of the models trained on problems from a single domain with the Neural Data Router, a neural baseline that represents the area of research on small-scale architectures specialized on solving systematic reasoning problems. We also benchmark the proposed systems trained in a multi-domain setting against OpenAI's GPT-4 and o1-preview, a general-purpose large language model and a recently proposed LLM-based architecture designed to solve complex reasoning tasks, respectively. The systems we propose consistently outperformed both the Neural Data Router when trained in a single-domain setting, and GPT-4 in their multi-domain version. While OpenAI's o1-preview demonstrated surprising capabilities in the solution of even non-trivial problems, the Neural Rewriting System consistently outperformed the model on the hardest problems we considered.

We then critically analyse the benchmarking results together with a detailed error analysis to answer the second research question, thereby assessing the impact of our design choices on the system performance. Finally, to address the third research question, we compare the time and space efficiency of the two systems and provide empirical evidence on the impact of our design choices on efficiency in terms of memory consumption, training and inference time.

In Chapter 5 we reconsider the questions posed in the beginning of the thesis and draw conclusions based on the pieces of evidence and arguments exposed in the previous chapters.

1.4 Publications

The contributions presented in this thesis appeared, in part or in total, in conference and workshop proceedings, whose peer-reviewed publication process has contributed to shaping the ideas we present here. Specifically,

- The evaluation framework and experimental results we propose to assess the impact of model scale and fine-tuning on large-language models capability to generalize systematically, which are presented in Chapter 3, were previously described in Petruzzellis, F., Testolin, A., Sperduti, A. (2024). Assessing the Emergent Symbolic Reasoning Abilities of Llama Large Language Models. In: *Wand, M., Malinovská, K., Schmidhuber, J., Tetko, I.V. (eds) Artificial Neural Networks and Machine Learning – ICANN 2024. ICANN 2024. Lecture Notes in Computer Science, vol 15020. Springer, Cham.* [35]

- The evaluation framework and experimental results on the impact of different specialized prompting methods for reasoning tasks on systematic reasoning capabilities of GPT models, which are presented in Chapter 3, previously appeared in Petruzzellis, F., Testolin, A., Sperduti, A. (2024). Benchmarking GPT-4 on Algorithmic Problems: A Systematic Evaluation of Prompting Strategies. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)* [36]

- A simple architecture incorporating some of the key concepts in the neuro-symbolic framework we propose in Chapter 4 was presented in Petruzzellis, F., Testolin, A., Sperduti, A. (2024). A Hybrid System for Systematic Generalization in Simple Arithmetic Problems. In *CEUR Workshop Proceedings of NeSy 2023, 17th International Workshop on Neural-Symbolic Learning and Reasoning, Certosa di Pontignano, Siena, Italy* [37]

- The Neural Rewriting System, one of the two implementations of the neuro-symbolic framework to learn convergent term rewriting systems we present in Chapter 4 and some of the related experimental results, were first described in Petruzzellis, F., Testolin, A., Sperduti, A. (2024). A Neural Rewriting System to Solve Algorithmic Problems. To appear in: *Proceedings of the 27th European Conference on Artificial Intelligence (ECAI 2024)* [38]

- The Fast Neural Rewriting System, the second implementation of the neuro-symbolic framework we present in Chapter 4, the experimental results on multi-domain training and the analysis on efficiency, are also described in Petruzzellis, F., Testolin, A., Sperduti, A. (2024). Learning Neuro-Symbolic Convergent Term Rewriting Systems. Unpublished manuscript in preparation.

Chapter 2

Background

The purpose of this chapter is to provide both historical and contemporary context for various approaches that contribute to the overall goal of designing neural and neuro-symbolic architectures capable of System-2 level reasoning. The concept of compositionality has been central to debates about the nature of cognition, learning, and reasoning, shaping both historical and contemporary research in artificial intelligence and cognitive science. This chapter explores the evolution of this debate, beginning with the classical dispute between connectionists and symbolic cognitivists, who diverged on whether the compositional nature of thought and language should be implemented in artificial systems in a purely symbolic or subsymbolic fashion. The discussion then shifts to recent advances in compositionality, presenting a theoretical framework, key benchmarks, and models that aim to capture this property in neural networks. Further, the chapter addresses the intersection of compositionality and learning algorithms, highlighting how formal languages embody compositional principles, thus linking this research area with that focused on compositionality. Finally, the chapter concludes by examining compositional reasoning and algorithmic learning through the lens of large language models, which have prompted the re-evaluation and exploration of these concepts in new ways.

2.1 An historical debate

2.1.1 The concept of compositionality

Among the aspects that have been considered foundational to human intelligence is compositionality. The concept is usually illustrated with the following sentence: "the meaning of a whole is a function of the meanings of the parts and of the way they are syntactically combined." One immediately sees that the concept concerns the formation of meaning and its expression with some kind of language. The sentence suggests that the interpretation of complex expressions is systematically determined by the interpretation of their simpler constituents and the rules used to combine them.

Compositionality is a property that can be identified in both natural and formal languages. Hierarchy is highlighted through the lenses of compositionality as a key common property of natural and formal languages. It refers to how parts are organized into struc-

tured layers, where smaller units (such as words or symbols) are nested within larger ones (such as sentences or formulas), each contributing to the overall meaning.

In natural languages, compositionality is manifest in the way words combine into phrases, phrases into clauses, and clauses into sentences, following syntactic rules. The hierarchical structure of natural language is most easily seen in syntax trees, where each level of the tree corresponds to a different unit of meaning. Natural language hierarchy also allows for recursion, where elements can be nested within other elements, such as embedding a clause within a clause: "The cat, which was very tired, sat on the mat." This recursive property is a hallmark of the hierarchy in natural languages, and it supports the infinite generativity of language—enabling speakers to construct and understand an endless variety of new sentences.

However, despite the hierarchical nature of natural language, there are differences in the way compositionality functions compared to formal languages. Natural languages can be more flexible, context-dependent, and ambiguous. For example, idiomatic expressions like "break a leg" are non-compositional in that the meaning of the whole phrase does not directly derive from the meanings of the individual words. Additionally, context plays a significant role in resolving ambiguity, as a word or phrase may take on different meanings depending on its environment.

In formal languages, such as mathematical logic or programming languages, every symbol has a fixed meaning, and the rules governing how symbols can be combined to form new meanings are strict and unambiguous. For example, in propositional logic, a well-formed formula like $P \wedge Q$ is composed of two propositions, P and Q, combined using the logical operator $\wedge$ (AND). The meaning of the formula as a whole is a function of the truth values of P and Q and the operator that connects them.

2.1.2 Two perspectives on human thinking

The concept of compositionality, particularly in the domain of linguistics, is closely tied to the work of Noam Chomsky. Chomsky's theory of generative grammar introduced the idea that the structure of language could be described through formal rules, enabling the generation of an infinite number of grammatically correct sentences from a finite set of components [39, 40]. This framework rested heavily on the principle that the meaning of complex expressions, such as sentences, arises from the meanings of their parts—words—and the way those parts are combined according to syntactic rules. However, while Chomsky's theories provided a strong foundation for linguistics, the rise of connectionist models in the 1980s challenged the dominance of symbolic representations and the explicit rule-based systems he proposed. This led to significant debates in cognitive science about the nature of language and how it is represented in the mind.

In the late 1980s, connectionist models gained attention as an alternative to symbolic approaches in modeling cognitive processes, including language [41, 42]. Unlike Chomsky's rule-based frameworks, connectionist models relied on distributed representations and patterns of activation across networks of simple, neuron-like units. These models aimed to explain cognitive processes, including language comprehension, by learning from data and adjusting the weights between units, instead of using predefined rules or structures.

This shift sparked a heated debate in cognitive science, particularly between Fodor and

Pylyshyn, who were critics of connectionism, and the proponents of neural network models. Fodor and Pylyshyn argued that connectionist models lacked the capacity to explain human-like compositional reasoning. Their central critique was that connectionist systems struggled to represent the structured, rule-governed nature of language and thought, which is inherently compositional. According to Fodor and Pylyshyn, symbolic systems are necessary to capture how humans can effortlessly combine familiar parts in novel ways, as required for both language and general reasoning.

The core of the debate between Fodor and Pylyshyn and the connectionists rests on the assumption that there is a fundamental overlap between thought and language. If language can be shown to follow compositional principles, and if human beings understand language through their cognitive processes, it stands to reason that human thought must also be inherently compositional. This view aligns with Fodor's Language of Thought Hypothesis (LOTH), which posits that thought itself has a language-like structure [23]. According to LOTH, mental representations are built from basic symbols that combine according to syntactic rules to form complex thoughts. These representations are compositional: just as the meaning of a sentence depends on the meaning of its individual words and their arrangement, the content of a thought depends on the structure of its components. Fodor and Pylyshyn maintained that without this compositional structure, it would be impossible for humans to reason about complex ideas, understand novel situations, or generalize from known experiences.

The connectionists, on the other hand, challenged the notion that human cognition needed to rely on such strict compositional rules. They argued that the mind might operate through more flexible, distributed processes, which do not require the explicit manipulation of symbols or the kind of syntactic structure Fodor and Pylyshyn assumed.

The debate about compositionality and the nature of thought, rooted in linguistics and sparked by connectionist models of language, significantly influenced the research directions of the connectionist community, especially in the study of how structure and symbols could be represented and processed within neural networks. As connectionists aimed to challenge the symbolic view of cognition, they proposed connectionist models that could process symbolic and structured information still using distributed representations. Notable works were BoltzCONS [43], a connectionist model developed to represent structures like trees or lists by incorporating mechanisms in the network to represent structured data in a distributed manner; Hinton's work on representing hierarchical structures and part-whole relationships in neural networks [44]; Smolensky's Tensor Product Representation, a mechanism for encoding variable bindings in connectionist systems by using tensor products [45].

2.2 New interest in compositionality

In recent years, there has been a resurgence of interest in the compositional reasoning capabilities of deep learning models, particularly in the context of small-scale neural networks. While the connectionist systems presented in the 1990s incorporated some aspects of compositionality by introducing ad-hoc architectural elements, modern research has also aimed to investigate whether contemporary general-purpose deep learning models, especially recurrent and transformer-based architectures, can learn to generalize compositionally.

2.2.1 Five properties of compositionality

In a comprehensive work on compositionality in deep learning systems, Dieuwke Hupkes and colleagues provided a valuable systematization of the different aspects of compositionality [46]. They outlined five key aspects that are important for understanding and testing the compositional abilities of neural models: systematicity, productivity, substitutivity, localism, and overgeneralization. These aspects help clarify which specific characteristics are being tested in neural networks, moving beyond the simple question of whether models are "compositional" or not. As we mentioned in the Introduction, the type of reasoning problems we consider in this thesis have strong compositional properties. This categorization can thus be useful to further characterize the problems we consider, as we will describe more thoroughly in Section 3.1, and to better define the capabilities of the models we test on them.

Systematicity refers to a model's ability to generalize across familiar parts in new ways. For instance, if a model understands phrases like "black cat" and "brown dog," it should also understand the novel combination "black dog." This capability reflects the idea that, rather than memorizing entire sentences, models should store the parts (like "black," "dog") and apply rules to combine them in new configurations, leading to new meanings. Systematicity is considered a hallmark of compositional reasoning because it shows how knowledge of individual components can be flexibly recombined.

Productivity describes the generative power of language and compositional systems noted by Chomsky, which allows a finite set of rules and elements to create an infinite number of combinations. As often described by the phrase "making infinite use of finite means," this aspect emphasizes that compositional systems, such as language, are not limited to the specific combinations they have encountered before. Therefore, models should be able to apply rules of combination to generate and understand infinitely long or complex structures.

Substitutivity is based on the principle that the meaning of a complex expression is determined by the rules used to combine its parts and the meaning of the individual components. This implies that substituting one part of a structure with another that has the same meaning should not change the overall meaning. For example, replacing "cat" with "feline" in the phrase "the black cat" should preserve the meaning. This aspect tests whether models can maintain compositional meaning when parts are replaced, focusing on the interchangeability of equivalent components.

While compositionality suggests that meaning is derived from combining parts according to rules, it does not specify how those rules should operate. **Localism** is a property of a symbolic system in which composition rulesare local, and therefore the meaning of a compound depends only on nearby elements. It contrasts with globalism, where context or broader structure may influence meaning. In language, both localist and globalist composition mechanisms co-exist. Testing for localism evaluates whether models apply composition rules that are applied strictly within a narrow scope, or if composition instead are influenced by the broader context.

Overgeneralization involves applying learned rules too broadly, including to exceptional cases where the rule does not actually apply. This is often seen in language learning, such as when children overapply rules for creating the past tense of verbs, leading to forms like "goed" or "breaked" instead of "went" or "broke." Overgeneralization is viewed as evidence of rule-based reasoning, as it demonstrates that the learner has internalized a rule but has

not yet learned its exceptions.

2.2.2 Benchmarking compositionality

To rigorously evaluate the compositional abilities of deep learning models, researchers have developed several benchmarks designed to test models' ability to handle compositional tasks. These benchmarks often focus on tasks where models need to generalize beyond the data they are trained on by combining learned components in novel ways.

The five principles of compositionality identified by Hupkes et al. [46] have also been embedded in the PCFG dataset [46], designed to test these capabilities in neural models. This dataset consists of syntactically generated sentences based on a probabilistic context-free grammar, with data splits specifically constructed to evaluate models' abilities to generalize compositionally across varying linguistic structures.

SCAN (Simple Compositional Arithmetic for Navigation) [47] is designed to test a model's ability to understand and execute instructions composed of simple actions. In this task, a model is trained to map commands (such as "walk twice and jump") to sequences of actions (e.g., ["WALK", "WALK", "JUMP"]). SCAN requires a model to translate unseen combinations of actions during testing. For example, while the model may have seen the command "walk twice" and "jump," it may not have seen them combined as "walk twice and jump." COGS (Compositional Generalization in Syntax) [48] is focused on the syntactic generalization capabilities of neural networks. COGS presents models with a variety of syntactic structures, such as subject-verb-object constructions, and tests whether they can generalize to unseen syntactic forms during evaluation. For example, a model trained on sentences like "The boy sees the dog" must generalize to more complex sentences like "The dog that the boy sees runs."

The Lookup Table Composition task [49], focuses on evaluating a model's ability to perform compositional mapping based on table lookup operations. In this benchmark, a model must learn mappings from input symbols to output symbols, with the challenge being that the model has to generalize to unseen combinations of inputs and outputs.

The CFQ (Compositional Freebase Questions) benchmark [50] was designed to evaluate compositional generalization in natural language understanding. CFQ is based on a dataset of natural language questions paired with structured queries (SPARQL) over a knowledge base. The key challenge of CFQ lies in its careful control of the compositional complexity of the questions, ensuring that while individual components of the queries (entities, relations) are familiar to the model, their combinations are novel at test time. Inspired by observations of the failures on CFQ, Csordás, Irie, and Schmidhuber [51] proposed CTL++, a variant of the Lookup Table Composition (CTL) benchmark, designed to test neural networks' ability to generalize systematically to unseen compositions of symbolic functions. While the original CTL focused on evaluating length generalization or productivity, CTL++ is specifically crafted to assess systematicity. The task involves compositions of unary symbolic functions, where functions are grouped, and the model is tested on novel compositions that were not encountered during training.

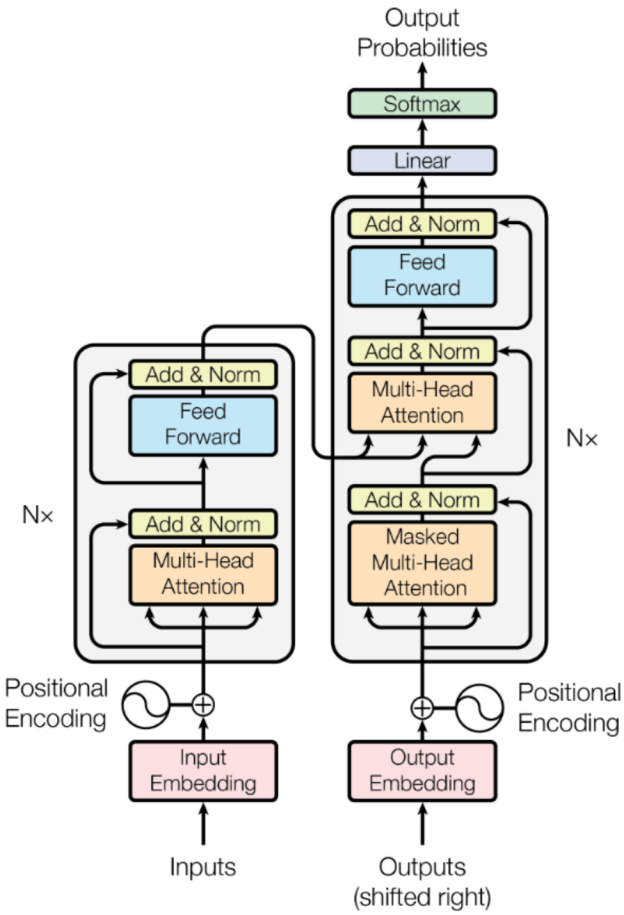


Figure 2.1: Schematic representation of the Transformer architecture. The two vertical blocks represent the encoder and the decoder. Figure is reproduced from Vaswani et al. [52]

2.2.3 Architectures for compositional reasoning

In addition to developing benchmarks, researchers have leveraged these datasets to evaluate the compositional generalization abilities of neural models. Many of these works were contemporary to the popularization of the transformer architecture [52] and study its capabilities, the impact of some architectural elements on compositional generalization, or extend it to enhance the generalization capabilities of the model. In the following section, we therefore provide the reader with a technical description of the transformer, including details on modifications to the self-attention matrix and positional encondings proposed in the literature, which will also be relevant background for our own work. Then, we provide an overview of the body of work studying the impact of architectural components and proposing novel methods to achieve compositional generalization in neural and neuro-symbolic systems.

The Transformer architecture

The Transformer architecture was introduced by Vaswani et al. [52] and quickly became central to the field of natural language processing. Its relevance can be traced back to earlier

breakthroughs in neural machine translation, particularly the introduction of the attention mechanism by Bahdanau, Cho, and Bengio [53]. The attention mechanism allowed neural networks to dynamically focus on different parts of the input sequence, addressing limitations in long-range dependencies. As attention's importance grew, the Transformer leveraged this mechanism fully by introducing the self-attention block, where each element of the input sequence attends to every other element.

We will now describe the main elements of the architecture in greater detail. Given a sequence of input tokens, these are first converted into a continuous vectors using an embedding layer, which maps discrete tokens to dense vectors of size d_{model}. Since the transformer lacks any recurrence or convolution, positional encodings are added to the input embeddings to provide information about the position of tokens in a sequence. The original transformer uses sinusoidal positional encodings:

$$\text{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \tag{2.1}$$

$$\text{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \tag{2.2}$$

where pos is the position of the token in the sequence, i is the dimension index, and d_{model} is the dimensionality of the model.

Once the input tokens are embedded, the model computes Q (query), K (key), and V (value) vectors by linearly projecting the embedded tokens into three different subspaces using learned weight matrices. For each token embedding x_i, the projections are:

$$Q_i = x_i W^Q, \quad K_i = x_i W^K, \quad V_i = x_i W^V \tag{2.3}$$

where x_i is the embedded representation of the i-th token, $W^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, and $W^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ are learned projection matrices, d_{model} is the dimension of the input embedding, and d_k, d_v are the dimensions of the projected spaces for queries, keys, and values respectively. The query, key and value row vectors are then stacked to obtain three matrices: $Q \in \mathbb{R}^{n \times d_k}$, $K \in \mathbb{R}^{n \times d_k}$ and $V \in \mathbb{R}^{n \times d_v}$.

The core component of the transformer is the scaled dot-product attention, which computes a weighted sum of values V using queries Q and keys K. Formally, it is computed as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V \tag{2.4}$$

where $Q \in \mathbb{R}^{n_q \times d_k}$ is the query matrix, $K \in \mathbb{R}^{n_k \times d_k}$ is the key matrix, $V \in \mathbb{R}^{n_k \times d_v}$ is the value matrix, d_k is the dimensionality of the key vectors, and n_q and n_k are the number of queries and keys respectively. The scaling factor $\frac{1}{\sqrt{d_k}}$ ensures that the dot products have suitable magnitudes, avoiding extremely large gradients.

To allow the model to jointly attend to information from different representation subspaces, several scaled-dot product attention mechanisms with different learned projections are applied in parallel, forming the multi-head attention block. For h heads, the output is computed as:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h) W^O \tag{2.5}$$

where each head is:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.6}$$

with learned weight matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$, $W^O \in \mathbb{R}^{h \cdot d_v \times d_{\text{model}}}$.

After the attention mechanism, each position in the sequence is passed through a position-wise feed-forward network (FFN), which is applied independently to each token. This consists of two linear transformations with a ReLU activation in between:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2.7}$$

where $W_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$, $W_2 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$, $b_1 \in \mathbb{R}^{d_{\text{ff}}}$, and $b_2 \in \mathbb{R}^{d_{\text{model}}}$.

The transformer architecture follows a multi-layer encoder-decoder structure (see Figure 2.1). The encoder consists of multiple layers, each composed of a multi-head self-attention mechanism followed by a position-wise feed-forward network. The decoder also has multiple layers, each containing three main blocks. First, the decoder applies masked multi-head self-attention, where a causal mask is used to ensure that each position can only attend to earlier positions in the sequence. This masking prevents the model from "seeing" future tokens, thereby enforcing the autoregressive property where each token is predicted based on previously generated tokens. Formally, the attention weights are set to zero for future positions, ensuring that token t_i can only attend to tokens $t_1, t_2, ..., t_i$. Next, the decoder applies multi-head encoder-decoder attention, where the decoder attends to the full output sequence of the encoder. This allows the decoder to incorporate contextual information from the input sequence, enabling it to generate tokens conditioned on the input. Finally, the decoder passes the output through a position-wise feed-forward network (FFN), similar to the encoder. Both the encoder and decoder layers are followed by layer normalization and residual connections. Unlike recurrent networks that process sequences iteratively, the Transformer processes input sequences in parallel through its self-attention block. This parallelism allows the architecture to take full advantage of modern GPU-based hardware, making it highly scalable. This is a key reason the Transformer has become the most widely used architecture in very-large-scale deep learning systems, such as large-language models (LLMs) and vision-language models (VLMs).

Despite its scalability, the Transformer's self-attention mechanism comes with high computational complexity, particularly scaling quadratically with respect to input length $\mathcal{O}(n^2)$. This quadratic scaling poses challenges as input sequences grow longer. Recent developments have proposed mechanisms for improving self-attention efficiency, including low-rank approximation [54] and sparse self-attention patterns.

Sparse Self-Attention Mechanisms In the standard self-attention mechanism, each token in the input sequence attends to every other token. Mathematically, this is captured by a self-attention matrix A, where each element A_{ij} represents the attention score between token i and token j. The matrix A is computed from the dot products of query (Q) and key (K) vectors as follows:

$$A_{ij} = \text{softmax}\left(\frac{Q_i \cdot K_j^T}{\sqrt{d_k}}\right) \tag{2.8}$$

which corresponds to the first term in the right-hand side of Equation 2.4.

Here, d_k is the dimension of the key vectors, and the softmax ensures that the attention weights sum to 1 for each token i. In the case of dense self-attention, A is an $n \times n$ matrix where n is the sequence length, resulting in quadratic complexity $O(n^2)$, as we previously described.

Sparse self-attention mechanisms have been proposed as a method to reduce the computational complexity of the self-attention block in Transformers. In this type of self-attention, certain elements of the attention matrix are masked, meaning that not all tokens attend to each other. For a given token i, only a subset of tokens (its immediate neighbors) are attended to, and all other tokens are masked out. This is achieved by setting the masked elements to a very large negative value, typically $-\infty$, before applying the softmax, which effectively nullifies their contribution to the attention weights:

$$A_{ij} = \begin{cases} \text{softmax}\left(\frac{Q_i \cdot K_j^T}{\sqrt{d_k}}\right), & \text{if } j \in \mathcal{N}(i) \\ -\infty, & \text{if } j \notin \mathcal{N}(i) \end{cases} \tag{2.9}$$

where, $\mathcal{N}(i)$ represents the set of tokens that token i is allowed to attend to. By masking tokens outside this neighborhood, we reduce the number of attention calculations for each token.

Various sparsity patterns have been explored in the literature. Fixed patterns maintain a constant structure across all inputs, such as attending only to neighboring tokens within a defined window. This approach can be seen in models like the Longformer [55], which utilizes a sliding window attention mechanism. In this pattern, each token i is allowed to attend to a fixed number w of neighboring tokens, $\mathcal{N}(i) = \{i - w/2, \ldots, i + w/2\}$, where w is the window size. This significantly reduces the number of attended tokens from n to w, where $w \ll n$. For each token i, instead of computing attention with every other token in the sequence, the computation is restricted to only w tokens within the sliding window around i. This masking results in a sparse self-attention matrix where most of the off-diagonal elements are masked. The sliding window mechanism reduces the computational complexity of self-attention: specifically, the complexity of computing attention for a single token is reduced from $O(n)$ to $O(w)$, as the token only attends to w tokens within its window. Therefore, the total complexity for the entire sequence becomes $O(n \cdot w)$ where n is the length of the sequence and w is the fixed window size. Since w is typically much smaller than n, this results in linear scaling with respect to the input length. Notice that the sliding window still allows for a gradual propagation of information across the sequence through successive layers, as each layer enables attention to neighboring tokens, effectively increasing the receptive field without requiring direct long-range attention in a single layer.

The benefits of sparse self-attention mechanisms extend beyond just computational efficiency: they can improve memory efficiency, by reducing the amount of memory required to store attention weights. By focusing on a limited set of relevant tokens, sparse self-attention can enhance model performance on specific tasks by minimizing noise from irrelevant tokens. Further, sparse attention patterns may provide better interpretability, as they highlight which tokens the model considers most important for its predictions.

Positional Encoding Schemes In Transformer architectures, positional encodings are crucial for providing information about the order of tokens in a sequence. However, tradi-

tional sinusoidal positional encodings can limit generalization to longer sequences than those seen during training. To address this limitation, several approaches have been proposed. One thing some of these methods have in common is their ability to express positional information in a relative manner, allowing the model to capture the relationships between tokens based on their relative distances or ordering, rather than relying on their absolute positions. For example, in Rotary Positional Encodings (RoPE) [56] each token's positional information is represented by a pair of angles that correspond to the token's position in the sequence. The self-attention mechanism is modified such that the input embeddings of tokens are rotated in the complex plane according to their positional encodings, facilitating the representation of relative positions.

Instead of using fixed positional encodings, AliBi [57] adds learnable linear biases to the attention weights computed during the self-attention mechanism. The biases are defined based on the relative distances between tokens, enabling the model to effectively modulate the influence of distant tokens dynamically. Randomized or Label-based Positional Encodings [58, 59], which we adopt in our architectures and describe in greater detail in Section 4.3.1, introduce variability in the representation of token positions by utilizing randomization techniques to generate positional information. Instead of adhering to fixed sinusoidal patterns or predefined positional encodings, this approach assigns each token a unique, randomly generated encoding that can vary across different training instances, allowing the model to explore a wider range of positional relationships during training.

Architectural elements and training strategies

A significant body of research has been dedicated to understanding how architectural elements and training strategies affect the compositional generalization abilities of existing neural models. This line of work typically focuses on analyzing known architectures, such as recurrent neural networks (RNNs) and transformers, to identify the factors that enable or hinder compositional reasoning. In general, these studies aim to uncover how architecture components, particularly positional encodings, but also attention mechanisms or memory units, and training techniques, such as curriculum learning, impact a model's ability to generalize systematically.

Newman et al. [60] investigate the impact of training LSTM and transformer language models with or without an End of Sequence (EOS) token on their ability to generalize to sequences of different lengths. In an oracle setting, where models are forced to generate sequences of a specific length during testing, the authors compare models trained with EOS and without EOS. Their findings reveal that the absence of EOS tokens leads to significantly better length-extrapolative performance, achieving up to 40% improvement over in the SCAN benchmark.

Csordás, Irie, and Schmidhuber [61] explore how revisiting simple model configurations can significantly enhance the systematic generalization capabilities of Transformer models (both vanilla and Universal [62] variant). By making adjustments to basic elements such as embedding scaling, early stopping, the use of relative positional embeddings, they achieve substantial performance improvements across multiple benchmarks, including SCAN, CFQ, PCFG and COGS.

Li and McClelland [59, 63] investigate the performance of causal transformers (also called

decoder-only transformers) on a set of algorithmic tasks, such as copying, sorting, and hierarchical compositions of these operations. They focus on improving the model's ability to generalize to sequences longer than those encountered during training. By replacing the standard positional encoding in transformers with a novel approach that uses arbitrary labels paired with items in the sequence, they achieve strong generalization to longer sequences. Ruoss et al. [58] identify in positional encodings a primary cause of poor generalization, even in models using relative positional encodings [64]. To overcome this, they introduce a new family of randomized positional encodings that simulate longer sequences by randomly selecting an ordered subset of positions that fit the sequence length. In a large-scale evaluation across 15 algorithmic reasoning tasks, their method improves Transformer models' ability to handle unseen sequence lengths, resulting in an average 12.0% increase in test accuracy.

Using a reasoning task based on Sudoku, Nam et al. [65] demonstrate that successful out-of-distribution generalization is achievable if the training set includes examples sampled from the entire distribution of simpler tasks. Furthermore, they find that models trained with absolute positional encodings struggle to generalize unless positional alignment is carefully managed. However, they show that suppressing sensitivity to absolute positions helps overcome these limitations, allowing the model to better generalize out-of-distribution.

Compared to our investigations presented in Chapter 4, this body of work is focused on finding more general solutions that can be applied to problems of various kinds, including some of those we mentioned in Section 2.2.2, while we focus on a more specific class of problems. However, this body of research contributed significantly to the design of Neural Rewriting Systems, for example, motivating the choice of positional encodings that we make in our models.

Novel architectures

As the reader will discover in Chapter 4, some of the architectures developed in the following body of work are similar in some aspects to the Neural Rewriting System: the use of recursive or iterative mechanisms in output generation, the design of ad-hoc architectural elements to address specific characteristics of the problems at hand, and the use of the symbolic form of the input as a key element to rely on for the solution of the problem are all traits that our work shares with some of the ones presented here. However, reflecting the variety of this research area, there are also relevant differences between these works and ours: some are distant in scope and motivation, such as those that take a perspective closer to cognitive science, while some others propose models that rely on fundamentally different assumptions, such as the Neural Data Router, which we will consider in greater detail and also take as a baseline for our experiments.

Lake and Baroni [66] provide evidence that neural networks can achieve human-like systematicity when optimized for compositional skills by introducing the metalearning for compositionality (MLC) approach. This method involves training neural networks through a dynamic stream of compositional tasks, guiding the models to balance systematicity and flexibility. The MLC method enhances compositional skills in several systematic generalization benchmarks such as SCAN and COGS. The results highlight how a standard neural network architecture, when optimized through MLC, can replicate human-like generalization capabilities in compositional reasoning.

Another group of architectures aims to achieve compositionality by incorporating iteration or recursion mechanisms, which enable models to handle complex structures and relationships by processing sequences or hierarchical data in a way that mimics iterative or recursive cognitive processes. Ontañón et al. [67] introduce iterative decoding as a novel approach to enhance transformer compositional generalization. This method serves as an alternative to traditional sequence-to-sequence (seq2seq) models, demonstrating improvements in compositional generalization on benchmarks such as the PCFG and Cartesian product datasets. Their findings reveal that seq2seq transformers often fail to learn effective iterations unless these iterations are explicitly modeled. Iterative decoding addresses this by breaking training examples into a series of intermediate steps that the transformer learns to handle iteratively. During inference, the model feeds the intermediate outputs back into the transformer as inputs, continuing this process until an end-of-iteration token is predicted, thus facilitating better handling of complex compositional structures.

Setzler, Howland, and Phillips [68] explore decode-side generalization using gSCAN, a synthetic benchmark for compositional generalization in grounded language understanding. They introduce Recursive Decoding (RD), a novel approach for training and utilizing sequence-to-sequence (seq2seq) models that targets this form of generalization. Instead of generating an entire output sequence in a single pass, RD trains models to predict one token at a time. During inference, the model's predictions are used to incrementally update the inputs (i.e., the external gSCAN environment), which are then re-encoded for subsequent decoding steps. This process breaks down complex, out-of-distribution sequence generation tasks into a series of incremental predictions, allowing the model to handle each step as if it were encountering familiar training examples, thereby improving generalization.

Deshpande, Chen, and Lee [69] address the challenge of enabling neural models to perform mathematical reasoning by proposing a strongly supervised recursive framework for traditional transformer architectures. Previous methods have struggled with generalizing to complex problems because they require models to derive final answers in a single step. Their approach overcomes these limitations by breaking down each problem into a sequence of intermediate steps, which are teacher-forced during training. During inference, the autoregressive model generates each intermediate step recursively, progressively building towards the final solution.

Csordás, Irie, and Schmidhuber [70] address the challenge of systematic generalization in Transformers, particularly in algorithmic tasks where these models often struggle to direct information effectively within the grid structure represented by Transformer columns. To improve the learning of useful control flow, they introduce key modifications to the Transformer architecture, including the copy gate and geometric attention. These enhancements are incorporated into their novel framework, the Neural Data Router (NDR). The NDR demonstrates 100% accuracy in length generalization on the Compositional Table Lookup [49] task, and achieves near-perfect accuracy on a simple arithmetic task and a new variant of ListOps [71] that tests generalization across computational depths. Considering the effectiveness of this architecture on the benchmarks on which it was evaluated, we have chosen to adopt it as a baseline representative of specialized small-scale neural models for algorithmic reasoning. In order to provide the reader with the necessary background information on this system, we describe it in greater detail in the following section.

The Neural Data Router

The purpose of the NDR is to model structured tasks that demand multi-step reasoning, such as algorithmic or symbolic tasks. NDR is designed to support hierarchical processing, ensuring that at each layer of the network, the necessary computations can be carried out, while preserving the results of prior steps when needed.

To achieve this, several design choices and architectural elements have been introduced in the architecture. First, in the NDR, weights are shared between layers, allowing all learned operations to be available at every layer of the architecture. This design ensures that each layer can perform any operation needed for the task at hand, regardless of its depth. Second, the architecture is designed with a number of layers that matches or exceeds the number of computational steps required for the problem. This ensures that the model can perform the required transformations in a sequential manner, where each layer or block of layers contributes to one step in the overall reasoning process. Third, to allow processing longer sequence than seen during training, the model employs relative positional encondings [64], optionally gating them with absolute ones, to let the model choose the most appropriate encoding during training. Fourth, the NDR incorporates a copy gate mechanism that allows information to pass unchanged between layers when necessary. This feature is particularly useful for ensuring that intermediate results, which do not need to be altered at a particular computation step, are preserved and passed forward to the next layer without distortion. To achieve this, the model modifies the transformer encoder block to allow skipping it entirely. Seeing the input of each transformer layer as a matrix of column vectors, the column i in layer t can be denoted as $h^{(i,t)} = H_{t,i} \in \mathbb{R}^d$ where d is the state size, and $H_t \in \mathbb{R}^{N \times d}$ denotes the states of all N columns in layer t. Using the copy gate, each column i in layer $(t+1)$ processes the input H_t similarly to regular Transformers, using the standard multi-head attention operation:

$$a^{(i,t+1)} = \text{LayerNorm}(\text{MultiHeadAttention}(h^{(i,t)}, H_t, H_t) + h^{(i,t)}) \tag{2.10}$$

$$u^{(i,t+1)} = \text{LayerNorm}(FFN^{\text{data}}(a^{(i,t+1)})). \tag{2.11}$$

However, the output is gated as:

$$h^{(i,t+1)} = g^{(i,t+1)} \odot u^{(i,t+1)} + (1 - g^{(i,t+1)}) \odot h^{(i,t)} \tag{2.12}$$

$$g^{(i,t+1)} = \sigma(FFN^{\text{gate}}(a^{(i,t+1)})) \tag{2.13}$$

where $g^{(i,t+1)} \in \mathbb{R}^d$ and FFN^{data} and FFN^{gate} are implemented as two-layer feedforward blocks.

When the gate is closed i.e. $g^{(i,t+1)} = 0$ in Eq. 2.12, the entire transformation is skipped and the input is copied over to the next layer $h^{(i,t+1)} = h^{(i,t)}$. The gate (Eq. 2.13) is parameterized as a function of the output of the self-attention (Eq. 2.10), such that the decision to copy or transform the input for each column depends on the states of all columns.

Fifth, the NDR introduces a modified attention mechanism dubbed geometric attention. The mechanism is designed to focus attention patterns on localized regions. By controlling the scope of attention, the model can focus on specific parts of the input or intermediate results, allowing for more targeted and precise information processing. Like in regular self-attention, given an input sequence, each input embedding $x^{(i)} \in \mathbb{R}^d$ is projected to key

$k^{(i)} \in \mathbb{R}^{d_k}$, value $v^{(i)} \in \mathbb{R}^{d_v}$, query $q^{(i)} \in \mathbb{R}^{d_k}$ vectors, and the dot product is computed for each key-query pair, as described in Eq. 2.8. In geometric attention, the dot product is followed by a sigmoid function to obtain a score between 0 and 1:

$$P_{i,j} = \sigma(k^{(j)\top} q^{(i)}) \tag{2.14}$$

which is treated as a probability of the key at position j matching the query at position i. These probabilities are finally converted to the attention scores $A_{i,j}$:

$$A_{i,j} = P_{i,j} \prod_{k \in \mathbb{S}_{i,j}} (1 - P_{i,k}) \tag{2.15}$$

where $\mathbb{S}_{i,j}$ denotes the set of all indices which are closer to i than j is to i, and when two indices have the same distance to i, the one which is to the right of i (i.e., greater than i) is considered to be closer, i.e.,

$$\mathbb{S}_{i,j} = \begin{cases} k \in \{1, ..., N\} \setminus \{i, j\} : |i - k| < |i - j|, & \text{if } i < j \\ k \in \{1, ..., N\} \setminus \{i, j\} : |i - k| \leq |i - j|, & \text{if } j < i \end{cases} \tag{2.16}$$

In addition, the diagonal elements are zeroed-out by setting $A_{i,i} = 0$ for all $i = 1, ..., N$.

Finally, a directional encoding is added to the self-attention product in Eq. 2.14. The encoding carries information about the relative position of each key-query token pair in a scalar value:

$$D_{i,j} = \begin{cases} W_{\text{LR}} h^{(i)} + b_{\text{LR}}, & \text{if } i \leq j \\ W_{\text{RL}} h^{(i)} + b_{\text{RL}}, & \text{if } i > j \end{cases} \tag{2.17}$$

where $h^{(i)} \in \mathbb{R}^d$ denotes the state at position i and $W_{\text{LR}}, W_{\text{RL}} \in \mathbb{R}^{1 \times d}$, $b_{\text{LR}}, b_{\text{RL}} \in \mathbb{R}$ are trainable parameters.

This directional information is integrated into the score computation of Eq. 2.14 as follows:

$$P_{i,j} = \sigma\big(\alpha\big(W_q h^{(i)} + b_q\big)^\top W_{k,E} h^{(j)} + \beta D_{i,j} + \gamma\big) \tag{2.18}$$

where the matrix $W_q \in \mathbb{R}^{d_k \times d}$ maps the states to queries, $b_q \in \mathbb{R}^{d_k}$ is a bias for queries, $W_{k,E} \in \mathbb{R}^{d_k \times d}$ maps states to keys, and $\alpha, \beta, \gamma \in \mathbb{R}$ are learned scaling coefficients and bias.

Neuro-symbolic architectures

Finally, a body of work has focused on addressing compositional generalization challenges through neuro-symbolic architectures, which combine the strengths of neural networks with symbolic reasoning. These approaches aim to bridge the gap between the flexible, data-driven

learning capabilities of neural networks and the rigorous, rule-based reasoning of symbolic systems. Within the broader area of research on neuro-symbolic integration, we focus here specifically on works that address compositional generalization problems.

Nye et al. [72] introduce a neuro-symbolic model designed to learn complete rule systems from a small set of examples. Rather than predicting outputs directly from inputs, their model is trained to induce the explicit rules governing a given set of examples, utilizing techniques from neural program synthesis. The authors demonstrate that their rule-synthesis method significantly outperforms traditional neural meta-learning techniques across three domains: an artificial instruction-learning domain that evaluates human-like learning, the SCAN challenge datasets, and the task of translating number words into integers across various human languages. Liu et al. [73] present a novel approach to compositional generalization by integrating memory-augmented neural models with symbolic functions, inspired by cognitive theories that suggest compositionality can be represented using variable slots and symbolic expressions. Their model features two complementary neural modules: Composer and Solver. The Composer handles the creation of symbolic expressions, while the Solver interprets and processes these expressions, reflecting the cognitive argument that compositionality involves structured symbolic reasoning. This setup allows the model to be trained end-to-end using a hierarchical reinforcement learning algorithm. Their experiments on the SCAN benchmark reveal that their model excels in compositional generalization, achieving 100% accuracy on all tasks previously tackled by other methods. Chen et al. [74] introduce the Neural-Symbolic Stack Machine (NeSS), a hybrid architecture where a neural network generates traces, which are then processed by a symbolic stack machine equipped with sequence manipulation capabilities. NeSS achieves 100% generalization performance across four distinct domains without requiring training supervision on execution traces. These domains include the SCAN benchmark for language-driven navigation tasks, few-shot learning of compositional instructions, compositional machine translation, and context-free grammar parsing tasks.

2.3 Learning Algorithms

Several of the works discussed earlier also evaluated their models on so-called algorithmic tasks like copying, sorting sequences, and mathematical reasoning in addition to compositional reasoning (see, for example, the tasks considered by Li and McClelland [63], the Mathematics dataset [75] in Csordás, Irie, and Schmidhuber [61], or the arithmetic task in Deshpande, Chen, and Lee [69]). Both research areas explore how neural networks can go beyond pattern recognition to handle more structured tasks that require systematic processing of data, whether it's in linguistic composition or solving algorithmic problems.

This connection between compositionality and learning algorithms is natural, as compositionality is a feature of both natural and formal languages. As previously noted, the key aspects of systematicity, productivity, substitutivity, and localism apply well to formal languages, just as they do to natural language. The principle of systematicity in formal languages ensures that if a model can solve one problem, it can systematically solve related problems by applying similar rules. Productivity allows for the generation of an infinite number of expressions from a finite set of rules, and substitutivity ensures that parts of a

structure can be replaced while preserving the overall meaning. In the case of local, rather than global, mechanisms for the formation of meaning, the interpretation of an expression can be derived by applying the mechanism to small, localized groups of elements.

One key difference, however, is that while compositional generalization research often focuses on benchmarks with inherently compositional samples with hierarchical structures, this is not always the case in algorithmic tasks. Studies on sequences of random vectors, for instance, do not necessarily involve hierarchical compositions [63, 76]. Yet, despite these differences, both research areas share a central concern with productivity. Just as compositional generalization emphasizes the ability to process hierarchies independently of their depth or complexity, algorithmic capabilities emphasize the importance of handling data structures of indefinite size. The ability to generalize beyond the length and size of the training examples—whether in language tasks or algorithmic reasoning—is a prominent common trait that links both lines of research.

2.3.1 Architectures

Memory-augmented neural networks

Early research on neural networks for learning algorithmic tasks concentrated on designing new architectures and mechanisms that could enable the networks to learn and execute systematic procedures. A major focus was on the use of external memories, which were introduced to help networks handle tasks requiring multiple steps and the retention of information across different stages of computation. These efforts aimed to imbue neural models with the ability to follow structured procedures, akin to the way algorithms are executed in traditional computer science. To evaluate whether these models could learn such systematic capabilities, they were often tested on relatively simple datasets, especially when compared to the more complex compositionality benchmarks we previously discussed. These tasks, closely aligned with the computer science tradition, included challenges such as copying sequences, sorting, and recalling items from memory.

In 2014, Graves, Wayne, and Danihelka introduced the Neural Turing Machine (NTM) [76]. Their work was among the first in the deep learning era to revisit the problem of learning algorithms with neural networks and proposed an influential solution. Their architecture combines neural networks with external memory resources and enables interaction through attentional mechanisms. This design mirrors the functionality of a Turing Machine or Von Neumann architecture, but with the added benefit of being differentiable end-to-end, allowing for efficient training via gradient descent. The authors demonstrate that this system can infer simple algorithms, such as copying, sorting, and associative recall, based solely on input-output examples. Notably, the model showed strong generalization to longer sequences of random vectors than those seen during training, although the memory size emerged as the primary limitation.

This architecture sparked a wave of follow-up research, where several works explored different ways of improving or extending the NTM model to address its limitations and enhance its algorithmic capabilities. The Differentiable Neural Computer (DNC), introduced by Graves et al. [77], builds on the NTM by coupling a neural network with an external memory matrix that functions similarly to the random-access memory in conventional com-

puters. The DNC can read from and write to this memory, enabling it to solve complex tasks like finding the shortest path between points and inferring missing links in graphs. Importantly, it demonstrated generalization to specific structures, such as transport networks and family trees, but there was limited evidence for out-of-distribution generalization in these experiments. The Dynamic Neural Turing Machine (D-NTM) [78] further enhanced the original NTM by introducing learnable memory addresses, making memory access more flexible. This architecture was tested on tasks like associative recall and copying, but the authors did not demonstrate the model's ability to generalize beyond the distribution of training data. Another development came from Csordás and Schmidhuber [79], who focused on improving the NTM's performance on both an arithmetic task and a question answering task through a series of architectural modifications. Their work sought to enhance the precision and efficiency of the NTM for these structured tasks. Finally, Zaremba and Sutskever [80] proposed a method for extending the NTM with generic external resources, incorporating discrete interfaces for input, memory, and output, which the model could interact with via reinforcement learning. They tested their model on algorithmic tasks such as copy, repeat copy, and reverse, and showed that the model could generalize to longer sequences on the copy task.

In addition to architectures designed for algorithmic learning, several memory-augmented models have been proposed for other tasks. Memory Networks [81] were introduced for question answering (QA), where long-term memory acts as a dynamic knowledge base to generate textual responses. The End-to-End Memory Network [82] refined this idea by training the memory system end-to-end, which increased flexibility and allowed its application to language modeling, other than QA. The Kanerva Machine [83] was inspired by Kanerva's sparse distributed memory and designed as a conditional generative model that could adapt quickly to new data through a robust distributed reading and writing mechanism. Finally, Sparse Access Memory (SAM) [84] tackled the scalability issues in memory-augmented neural networks, demonstrating faster operations and more efficient use of physical memory while handling tasks requiring extensive time steps and memory capacity.

Other architectures

A number of other architectures have been developed to achieve systematic reasoning capabilities on mathematical and algorithmic problems, aiming to achieve strong generalization beyond the training data. The ones we consider here are focused on learning to solve mathematical problems. Indeed, the research on learning mathematics with neural networks intersects significantly with the work on learning algorithms, as both fields often explore systematic problem-solving abilities. Within the broader area of learning mathematics, some studies emphasize the development of models capable of solving tasks systematically, resembling algorithmic learning, while others focus on broader mathematical problem-solving without necessarily enforcing systematic procedures [85].

NeuralGPUs [86] address the limitations of Neural Turing Machines (NTMs) by being parallel and easier to train. They excel in tasks like long addition and multiplication, demonstrating the ability to generalize from short training instances to much longer ones. PointerNetworks [87] are designed for tasks where the output sequence elements correspond to positions in the input sequence, such as sorting and combinatorial optimization. Using

attention mechanisms, these models can generalize to variable-length inputs and have been tested on problems like the Travelling Salesman Problem.

NALUs (Neural Arithmetic Logic Units) [88] enhance networks by incorporating primitive arithmetic operations controlled by learned gates. They improve numerical generalization across various tasks, including time tracking, arithmetic over images, and translating numerical language into real values. Neural Arithmetic Units (NAUs and NMUs) [89] introduce components capable of exact addition, subtraction, and multiplication. The NMU, in particular, is notable for being able to learn to multiply vector elements, providing new capabilities in performing arithmetic tasks.

2.3.2 Neural Algorithmic Reasoning

Finally, a recently emerged research area on learning algorithms is Neural Algorithmic Reasoning [90]. This stream of research, first opened by Petar Veličković and collaborators, aims to train neural networks, specifically Graph Neural Networks (GNNs), to learn and execute classical algorithms on graph-structured data. The goal is to create models capable of reasoning systematically over data structures like graphs, generalizing well to new instances and achieving algorithm-like precision in tasks such as shortest path finding, graph traversal, and sorting [91].

Neural Algorithmic Reasoning leverages Graph Neural Networks (GNNs) due to a key advantage: their structure can be aligned with the computational steps of many common algorithms, a property referred to in the literature as "algorithmic alignment" [92]. This alignment makes GNNs particularly suitable for learning dynamic programming algorithms, which often operate over structured, graph-like representations of data [93]. One critical inductive bias in GNNs is their permutation invariance [94], which enables the model to learn algorithms where the input order does not affect the output—a property vital for certain algorithmic tasks. However, recent research has also begun exploring alternative inductive biases better suited for learning algorithms that involve sequential inputs, thus expanding the range of algorithmic tasks that GNNs can effectively solve [95]. Furthermore, when paired with Transformer-based large language models (LLMs), GNNs have demonstrated to improve their out-of-distribution generalization capabilities [96].

2.4 Compositional and algorithmic reasoning in LLMs

Large Language Models (LLMs) have represented a paradigm shift in Natural Language Processing and Artificial Intelligence more generally. We start by describing the key technical elements that allowed this paradigm shift, including the architectural innovation introduced by the transformer, training paradigm and scaling paradigm such as unsupervised pre-training and in-context learning through prompting.

2.4.1 The development of Large Language Models

The development of Large Language Models has been enabled most importantly by the transformer architecture and its core self-attention mechanics, which we described extensively in

Section 2.2.3. The transformer represented a breakthrough by enabling models to process entire sequences of data in parallel, unlike recurrent networks which process inputs sequentially. This shift allowed transformers to fully leverage GPU-based hardware, which excels at parallel computation, unlocking unprecedented scalability for training on large datasets. The self-attention mechanism within transformers, particularly the multi-head attention mechanism, empowered models to capture long-range dependencies in data more effectively, a crucial capability for natural language understanding tasks.

Scaling LLMs also relied heavily on innovations in computational strategies. Model parallelism and data parallelism became essential to distribute the immense computational workload of training larger models across multiple GPUs or TPUs. Additionally, the use of lower precision or mixed precision training reduced memory requirements without sacrificing model performance. This enabled more efficient training and allowed for scaling models to billions of parameters.

Another critical development was the shift from training specialist models in a supervised manner on domain-specific datasets to training generalist models on vast, multi-domain datasets [97]. Unsupervised pretraining paradigms, such as autoregressive language modeling, popularized by models like BERT [7] and GPT-2 [98], became a dominant approach. These models were pretrained on massive corpora of diverse text, allowing them to learn generalized language patterns without the need for manually labeled data. Following pretraining, these models could be fine-tuned on specific tasks using smaller, labeled datasets, a process known as transfer learning. This strategy vastly improved the ability to adapt models to various tasks, eliminating the need to train each model from scratch for every new task.

The presentation of scaling laws for language models, as articulated by Kaplan et al. [99], provided empirical evidence for the benefits of larger models, showing that model performance improves predictably with increases in data size, model parameters, and computational resources. This insight guided research towards scaling models as a primary avenue for advancing capabilities, leading to the creation of models like GPT-3 [13].

A final shift in the development of LLMs has been the movement towards few-shot and zero-shot learning through prompting techniques. Instead of requiring task-specific fine-tuning, models like GPT-3 demonstrated the ability to generalize across a wide range of tasks with minimal task-specific data, often simply by providing a few examples or instructions in the form of text prompts. A further advancement in the evolution of Large Language Models came with instruction fine-tuning, as introduced by Ouyang et al. [100]. This method involves fine-tuning through Reinforcement Learning through Human Feedback (RLHF), a technique explicitly designed to shift the distribution of models outputs towards those preferred by humans, making them more aligned with user expectations and capable of responding more effectively to a wide range of natural language prompts.

Thanks to these technical advancements, LLMs have achieved impressive results across numerous areas of natural language processing and opened up new avenues of exploration across various domains. As a result, many of the research interests previously discussed, from compositionality to algorithm execution and broader reasoning tasks, have increasingly involved LLMs. Notably, due to the fundamentally different nature of these models compared to small-scale deep learning models, it is not possible to evaluate these models in truly out-of-distribution scenarios (often, the training distribution is unknown or cannot be precisely

described). What is thus commonly done to evaluate their systematic reasoning capabilities is benchmarking how robust their performance is to increase in problem complexity, as defined depending on the type of problem considered. We now consider several results in this area, including specialized prompting techniques to improve reasoning in LLMs, as well as studies on their compositionality and algorithmic reasoning capabilities.

2.4.2 Eliciting reasoning via prompting

In recent years, several prompting methods have been developed to enhance the reasoning capabilities of large language models. "Reasoning," in the context of language modeling, often refers to question-answering tasks that require models to apply either commonsense knowledge or more specialized knowledge such as mathematical reasoning to derive accurate answers. These reasoning abilities have been assessed using large-scale benchmarks such as Winogrande [101] and HellaSwag [102] for commonsense reasoning, as well as more specialized tasks like Abstract Algebra from the MMLU dataset [103] and DM Math from The Pile [104]. Despite the impressive scale of recent models, scaling alone has proven insufficient to significantly improve reasoning performance [105]. Therefore, researchers have focused on developing specific prompting techniques aimed at improving the models' ability to reason. Below, we review some of the most impactful prompting methods designed to enhance reasoning performance in large language models.

One of the most well-known prompting methods for improving reasoning in large language models is Chain-of-Thought (CoT) prompting, introduced by Wei et al. [106]. This method involves providing examples of reasoning chains in context, prompting the model to generate intermediate reasoning steps before arriving at a final answer. This step-by-step approach mimics human problem-solving and has been shown to significantly enhance performance on tasks requiring multi-step reasoning, such as math word and commonsense reasoning problems. Building on this, Zero-shot Chain-of-Thought prompting [107], adapts the same idea but without providing any examples in context. Instead, the model is simply asked to reason "step-by-step" to solve the given problem, in a zero-shot fashion. Remarkably, this approach enables models to generate reasoning chains similar to those obtained with CoT prompting and improve performance on reasoning tasks without needing any prior task-specific examples. Yet another influential prompting method is Tree of Thoughts [108]. Unlike Chain-of-Thought, which follows a linear reasoning path, Tree of Thoughts expands the reasoning process by allowing the model to explore multiple reasoning paths at each step, effectively creating a tree structure. The model then evaluates and selects the most promising paths at each branch. This approach enables a more flexible and robust reasoning process, particularly for complex tasks that require considering multiple solutions or strategies simultaneously, enhancing performance across a variety of reasoning tasks.

Another notable general-purpose method is Self-consistency prompting [109]. This technique further improves reasoning by generating multiple reasoning paths for the same question and selecting the most consistent answer across these paths. By relying on the idea that reasoning consistency can lead to more accurate solutions, this method reduces the risk of errors introduced by relying on a single reasoning chain, leading to more reliable results.

2.4.3 Compositionality in LLMs

When investigating the compositional capabilities of large language models (LLMs), the term "compositionality" is interpreted in two distinct ways within the research community. One interpretation focuses on the ability of LLMs to integrate multiple pieces of information, processing them in several reasoning steps before producing an output. This type of compositionality is often benchmarked through multi-hop or complex question answering tasks, where models must handle intricate logic and layered reasoning. The second interpretation, rooted in cognitive science and AI, aligns more closely with systematic reasoning, where compositionality is akin to formal, rule-based human reasoning. To evaluate this, synthetic benchmarks with controlled characteristics are typically used, offering a more precise measure of systematic generalization.

Among those that investigate compositionality as a broader "reasoning capabilities" in complex question answering problems, Press et al. [110] introduce the concept of a "compositionality gap," which measures how often models can answer sub-problems correctly but fail to generate the overall solution. They find that as model size increases in the GPT-3 family, the single-hop performance improves faster than multi-hop reasoning, indicating that larger models are better at recalling facts but not at composing them. They also demonstrate that techniques like chain-of-thought (CoT) prompting help narrow this gap by enabling more explicit reasoning.

Given the widespread adoption of LLMs in many fields of application, several methods also attempt to improve the compositional reasoning capabilities of language models. Zhang et al. [111] tackle LLMs' compositional generalization issues in complex question-answering tasks with cumulative errors. They propose a human-guided tool manipulation (HTM) framework to generate and integrate subtools for specific subquestions, enhancing overall performance. Wang et al. [112] introduce the LOIRE framework, which aims to improve LLM performance by generating inferential rules that serve as scaffolding for reasoning tasks. The challenge of scaling inferential rule generation is addressed by leveraging symbolic logic, allowing the model to better handle complex reasoning scenarios. Li et al. [113] examine compositional reasoning failures in LLMs, finding many errors arise from generated implicit reasoning. They introduce CREME, a method for editing multi-head self-attention (MHSA) modules to correct errors and enhance reasoning performance.

In line with the second interpretation of compositionality, which aligns with the cognitive science and AI traditions, several works have also explored the systematic reasoning capabilities of LLMs on more formal, controlled benchmarks. Dhar and Sogaard [114] investigate the impact of model scaling and instruction tuning on compositional learning, using grammar-based benchmarks inspired by Hupkes' analysis [46]. They find that while scaling LLMs enhances their compositional reasoning abilities, instruction tuning can have the opposite effect, potentially hindering systematic reasoning. Dziri et al. [115] assessed transformer LLMs on tasks requiring structured, algorithmic reasoning, including multi-digit multiplication, logic grid puzzles, and dynamic programming. The authors reveal that, rather than developing true systematic problem-solving abilities, LLMs tend to reduce complex multi-step tasks into simpler linear subgraph matching, which limits their capacity for deeper compositional reasoning. Chen et al. [116] propose a prompting method, SKiC (skills-in-context), which presents foundational skills and compositional examples within the same

prompt. Benchmarks include tasks such as last-letter concatenation, arithmetic, and dynamic programming. The method significantly improves the LLM's ability to systematically generalize, even with minimal examples.

2.4.4 Algorithmic Reasoning in LLMs

Several works have explored the application of large language models to algorithmic tasks, focusing on their ability to process formal inputs and execute structured procedures. These studies target tasks in domains like computer science, mathematics, and logic, where models are required to follow explicit rules or computations. This line of research aims both to evaluate, understand and enhance LLMs' performance on these tasks.

Zhou et al. [117] propose a framework to understand the length generalization capabilities of transformers in algorithmic tasks. They introduce the RASP-Generalization Conjecture, which posits that transformers generalize better if a task can be solved by a short RASP program [118] that works for all input lengths. They demonstrate that tasks like parity and addition, which fall outside this group, are harder for transformers to generalize, but their insights lead to improved generalization performance on these tasks. Anil et al. [119] explore the performance of transformer-based language models on length generalization tasks such as parity and boolean variable assignment. The authors find that finetuning alone is insufficient for generalization across different input lengths. However, combining pretrained models with in-context learning and using scratchpad prompting (where intermediate steps are output before the final answer) significantly improves length generalization. Saparov et al. [120] evaluate the ability of LLMs to generalize in formal deductive reasoning tasks. Using a synthetic dataset that controls for deduction rules and proof complexity, they examine models' depth, width, and compositional generalization. Although LLMs demonstrate compositional generalization to more complex proofs from simpler ones, they struggle when faced with significantly longer proofs, indicating limitations in their reasoning capabilities.

Prompting methods for algorithmic problems

In Least-to-Most prompting [121], the model is guided to solve complex problems by breaking them down into a sequence of simpler sub-problems, which are solved progressively from least to most complex. This method has shown effectiveness in symbolic reasoning tasks like last letter concatenation and the SCAN benchmark, enhancing systematic generalization. Prompting with Scratchpads [122] requires the model to write out intermediate algorithms execution steps in a specific format. Tested on tasks like addition, polynomial evaluation, and executing Python programs, this method has demonstrated improved performance on algorithmic reasoning by allowing the model to explicitly process formal reasoning steps. Prompting with Self-Notes [123] encourages the model to take intermediate notes which, unlike Chain-of-Thought and Scratchpads, are not part of the model output. This method has been applied to symbolic reasoning tasks that involve evaluating short computer programs, improving the model's ability to track logical steps and reasoning paths. Algorithmic Prompting [124] focuses on teaching models to apply specific algorithms through in-context learning. Tested on tasks like long parity, addition, multiplication, and subtraction, this method structures prompts to teach models not only to learn individual algorithmic skills

but also to combine and apply them in new contexts, resulting in significant performance improvements on algorithmic reasoning tasks.

Chapter 3

Testing for systematic reasoning in deep architectures

In this chapter, we will present several original results that stem from the observations and research trends discussed in the previous chapters. Drawing on insights from the literature, we propose novel systematic generalization tasks focused on the simplification of nested mathematical formulas across various domains. These tasks are designed to probe the reasoning capabilities of both small- and large-scale neural architectures, aligning with two major areas of investigation outlined earlier: algorithmic reasoning with specialized small-scale models, and the broader reasoning capabilities of large language models.

The motivation behind these investigations is grounded in key questions raised by prior research. In particular, we aim to explore the limits of generalization for small-scale neural architectures that have demonstrated promise in algorithmic reasoning. On the other hand, large-scale language models, which have garnered much attention for their flexible reasoning abilities, are examined with an eye toward systematic reasoning. We aim to answer the following research questions: Do these models truly learn to reason systematically? Does this capability improve as model scale increases? Moreover, How reliably can systematic reasoning be elicited through prompting techniques?

The chapter is structured as follows: we will begin by introducing the novel problems we designed to investigate these questions, followed by a presentation of the models selected for analysis, their evaluation methodologies, and the results obtained. Finally, we will discuss the findings, connecting them to the research questions outlined above, and offer conclusions that shed light on the capabilities and limitations of current neural architectures in systematic reasoning.

3.1 Mathematical formulas as reasoning benchmarks

In this chapter, we focus on benchmarking small- and large-scale neural networks on the simplification of nested mathematical formulas. In these formulas, each operand can itself be another formula, thus representing an example of hierarchichy, a central concept in compositionality. This hierarchy implies that solving any given formula requires iterative reasoning steps, where sub-formulas are systematically simplified and substituted back into the main

expression. For example, the arithmetic expression `(12+(3-(4+5)))` can be solved by first identifying a solvable sub-expression, i.e. `(4+5)`, and then substituting the result obtained by solving that sub-expression, namely `9`, into the original expression, obtaining a simpler expression, i.e. `(12+(3-9))`, which can be further simplified by iteratively applying the same procedure.

This framework naturally embodies several core aspects of compositionality we described in Section 2.2.1, based on the work of Hupkes et al. [46]: systematicity, as the simplification procedure remains consistent across formulas of varying complexity. The same general rule can be applied at any level of nesting, allowing for systematic generalization from simpler to more complex structures. Productivity, since the recursive nature of the problem, where operands can themselves be formulas, supports arbitrary nesting. Substitutivity, which is inherent in the mechanism of reducing sub-expressions and replacing them within the larger formula without altering the overall structure. Localism, as the local impact of the substitutions on semantics is ensured by the strict interpretation rules of formal languages, as we discussed in Section 2.1.1.

The benchmark we propose, centered on the simplification of nested symbolic formulas, is highly relevant for evaluating both specialized small-scale models and Large Language Models (LLMs). Its design draws from existing benchmarks in compositional reasoning, but it extends them in ways that are valuable for testing the generalization capabilities of a wide range of architectures. For small-scale models, this benchmark is aligned with several established tasks that test neural networks' ability to generalize compositionally, such as the Compositional Table Lookup (CTL) [49], simple arithmetic tasks considered in [70], and the well-known ListOps dataset [71], a variant of which we consider in this work. For Large Language Models (LLMs), the proposed benchmark is equally relevant, as it captures the essence of symbolic reasoning tasks commonly used to assess their reasoning capabilities. Problems such as last letter concatenation or coin flip are often used in LLM research to test whether these models can follow an algorithmic reasoning process that involves multiple steps [107]. The nested formula simplification task aligns with these in spirit. Just like last letter concatenation requires LLMs to extract specific parts of a string and recombine them, the benchmark we propose requires models to simplify formulas by recursively applying a set of rules. Moreover, tasks that require algorithmic reasoning procedures, such as simplifying a formula or solving a complex arithmetic expression, are considered some of the hardest challenges for LLMs. The proposed benchmark offers a rigorous framework for testing whether LLMs can discover and apply these procedures consistently to formulas of varying complexity levels.

The problem framework allows for the generation of synthetic formulas across various domains, characterized by two key parameters: Nesting (the maximum depth of any operation within a formula) and Operands (the number of inputs each operation takes). By varying the values of the Nesting and Operands parameters, we could define an arbitrary number of *data splits* for each task, each featuring a different level of difficulty.

Since the complexity of formulas can be characterized in terms of two parameters, that is, the nesting level and the number of operands involved, the reasoning abilities of the models can be analyzed in a finer-grained way compared to other reasoning benchmarks. This allowed us to compare the performance of models of different sizes on problems of varying levels of difficulty and thus investigate symbolic reasoning abilities.

	2 Operands	3 Operands	4 Operands
Nesting 2	((-21+47)* (38*-69))	(-73- (33*54)+55)	((-28+32)-(28-11+65)+(13+53)- (-15*20))
Nesting 3	(57*((5+1)+ (-79+60)))	(((35-2+12)- 94+(62*- 30))+((-97*- 75)-(-10*- 53)+9)-74)	(((-6-41-91-80)-(-31*-22)- (-54*84)-(0+77))+((-77-27)-77- 86-96)+(91+20+(-3+3-30)+(-41- 65+6+89))-((-83-23+50)+34-(-93+4- 15-8)-(35*-26)))
Nesting 4	(-35*(((27* 53)+(-43*- 51))+ ((-19*81)+ (42*66))))	((((-86+25)- (-87+76-8)- (17-93+19))+ ((-22-79- 17)+72+4)+ (-80-(-96*-15)- 64))-32-36)	(((-66+(-52*51)+43-(-62+69+81+38))- ((97*83)+86-41-85)-((91+8+89)+ (-15+33+99)+12+(-6-53+18))- (-48-(64+77+36+69)+(-56+12- 80)-27))+(((-74+7)+(49+96-4)- (20-1)-(72-5-78))-(16+69+(59- 61+80+9)+(78+60+3))-(46+(19+10- 48+14)+(61*-4)+(0+86+40-4))+ (-53-79+(31*-94)-68))+(-16+81+71+ (-55-41+(-12*-73)-32))+(84-74+((13- 27+17-90)-(15+75+93)+(54+37- 62)+(71-23+46-4))-((61+14)-(-32- 87)+(68-22-25)-(14*-7))))

Table 3.1: Examples of inputs of the arithmetic task for the nine data splits characterized by parameters Nesting and Operands varying bewtween 2 to 4. The parameters regulate the complexity of formulas.

In our experiments, we consider formulas from three domains with different levels of complexity: expressions on lists of integers derived from the ListOps dataset [71], arithmetical expressions [37] and simple algebraic expressions. We describe each domain in more detail below.

3.1.1 ListOps

The ListOps dataset [71] was introduced as a simple benchmark to assess the capacity of neural networks to evaluate nested expressions with parse trees of different depths. The original dataset included formulas composed of four operations—minimum, maximum, median, and sum modulo 10—applied to lists of single-digit integers, where the final solution is always a single-digit integer. For example, one instance of the problem might be `[MIN[MAX2567][SM1293]13]`, drawn from a data split parameterized by $(N = 2, O = 4)$, with a unique solution of `1`.

To adapt this dataset to the problem framework we propose, we made it possible to specify both the number of operands appearing in each operation and the operations' nesting depth. We also simplified the task by using only three operations: minimum, maximum, and sum modulo 10.

3.1.2 Arithmetic

For the arithmetic task, we generated expressions involving sum, subtraction, and multiplication operations between integers sampled from the range (-100, 100). The goal was to test the ability of models to systematically execute a sequence of operations rather than their mathematical competence with multi-digit numbers [125]. To maintain manageable problem complexity and focus on recursive solution steps, each intermediate value was computed using modulo 100, following previous work on systematic generalization in transformers [70].

In Table 3.1 we report, as a reference, examples of formulas from the arithmetic task that have been drawn from the nine data splits which result from taking the values in the Cartesian product of the sets $N = \{2, 3, 4\}$ and $O = \{2, 3, 4\}$, representing values of the Nesting and Operands parameters, respectively.

3.1.3 Algebra

We finally adopted the general problem structure to the domain of symbolic mathematics, considering a subset of algebraic expressions in which all formulas can be reduced to a minimal form, i.e. either a single number, a monomial, or a binomial. We automatically generate algebraic expressions consisting of sums and subtractions between monomials. Each monomial can contain up to four variables and has a numerical coefficient in the range $(-100, 100)$. For example, the formula $(((30xy + 33xy) + (-80xy + 62xy)) - 62xy)$ is sampled from the data split parameterized by $(N = 3, O = 2)$ and its simplified form is $-17xy$.

3.2 Systematic generalization in the Neural Data Router

Building on prior research highlighting the potential of small-scale networks in algorithmic reasoning, we focus on the Neural Data Router as a representative model. We systematically increase task complexity—by varying formula nesting depth and the number of operands—to test its ability to generalize beyond simpler cases. Our experiments assess how the model's performance scales with complexity, measuring its effectiveness at different problem difficulty levels and identifying the point at which generalization begins to falter.

3.2.1 Model

The Neural Data Router [70] has recently been proposed as a modification of the Encoder module in the Transformer architecture [52] with the specific goal of solving problems where applying the same resolution step iteratively (such as solving a sub-expression in a complex formula) can lead to the final solution. For a detailed technical description of this architecture, we address the reader to Section 2.2.3, where we first described the model.

The model was originally tested on three tasks: ListOps [71], Compositional Table Lookup (CTL) [49], and arithmetic formulas with single-digit operands and intermediate values taken modulo 10. In the original work, the final result could be collected in the first or last position of the encoded sequence, since these problems always have single-digit integers as targets. This no longer holds in the algorithmic problems we consider which can

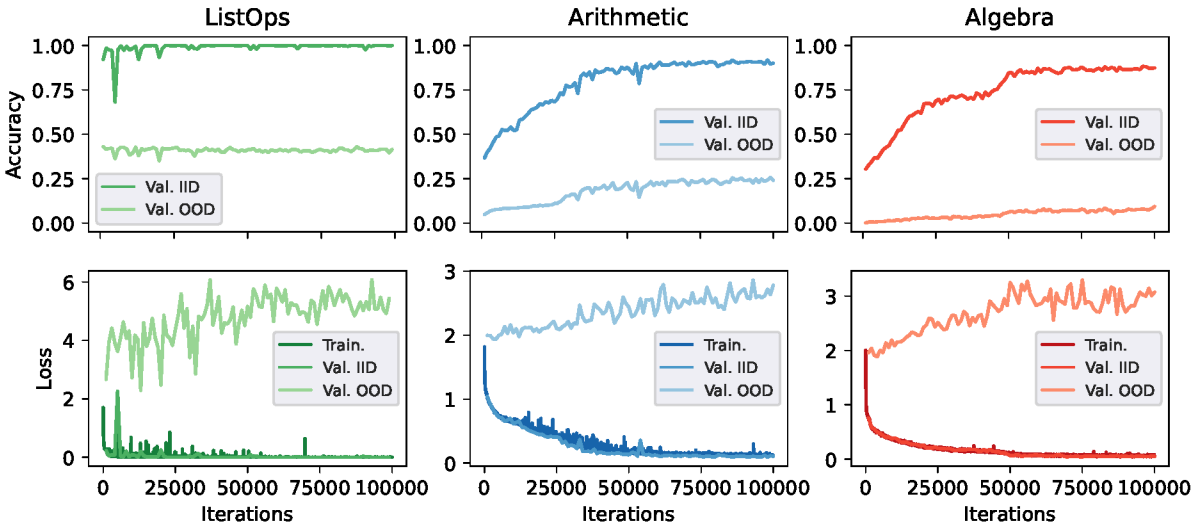


Figure 3.1: Accuracy and loss of the Neural Data Router during training on the three algorithmic tasks. Val. IID and Val. OOD refer, respectively, to in-distribution and out-of-distribution validation sets, described in Section **??**. The model overfits the in-distribution split on all tasks, failing to generalize to more difficult samples.

have a solution containing more than one token. We thus adapted the model by modifying the mechanism of collection of the final result, considering a window at the beginning of the final sequence produced by the encoder which is as large as the expected target.

3.2.2 Training details

We train and evaluate the Neural Data Router on ListOps, arithmetic and algebraic formulas. For all tasks, we include in the training set examples drawn from the data defined by the parameters values $(N = 1, O = 1)$ (i.e., results), $(N = 1, O = 2)$, $(N = 2, O = 2)$ and $(N = 2, O = 3)$. The training sets are balanced across the four different data splits and include $400,000$ samples for all tasks. We select the model's hyperparameters reproducing the search procedure described in the original work. We define two validation sets: an in-distribution validation set which mirrors the composition of the training set, and an out-of-distribution validation set which includes samples from more difficult splits, represented by the parameters $(N = 2, O = 4)$, $(N = 3, O = 2)$, $(N = 3, O = 3)$, $(N = 3, O = 4)$, $(N = 4, O = 2)$, $(N = 4, O = 3)$ and $(N = 4, O = 4)$. Both validation sets are balanced across data splits and include $1,000$ samples per data split, following the same model selection protocol used in the original work. We select the best performing model on the out-of-distribution validation set applying the Bayesian hyperparameter search tool provided by the Weights and Biases MLOps platform[1] to optimize the following hyperparameters: learning rate, number of encoder layers, dimensionality of the hidden state, number of heads, weight decay, dropout, attention dropout and size of the hidden feed-forward layer. The parameter

[1] https://wandb.ai

ranges used in the hyperparameters search procedure are the same reported in the original work.

We test the model on the same nine data splits for all three tasks, namely the ones that result from taking the values in the Cartesian product of the sets $N = \{2, 3, 4\}$ and $O = \{2, 3, 4\}$, representing values of the Nesting and Operands parameters, respectively. It should be noted that the scenarios in which we probed this model can be considered more challenging than the ones considered in the original work. Indeed, we used a smaller training set for the ListOps task (the original training set included formulas with nesting depth up to 5), and we considered more complex arithmetic expressions, as well as algebraic expressions on which the model was never tested before.

3.2.3 Results

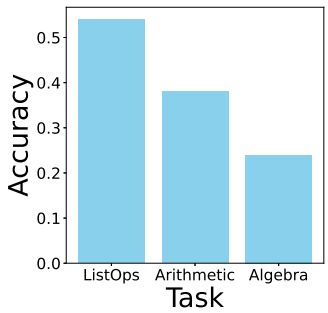


Figure 3.2: Global average performance of the Neural Data Router. Values represent output accuracy on the test set in percentage.

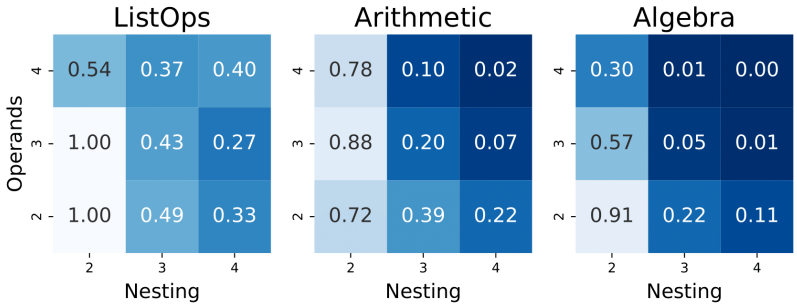


Figure 3.3: Average performance of the Neural Data Router by data split. Values represent output accuracy on the test set in percentage: the performance clearly decreases on data splits of higher complexity. Each data split contains 100 samples.

Figure 3.1 shows how accuracy and loss of the best Neural Data Router configuration resulting from the hyperparameters search evolve during training. It is interesting to note that while the performance on the in-distribution validation set grows steadily, reaching more than 85% accuracy on all tasks, the performance on the out-of-distribution validation set grows much more slowly, becoming almost constant when learning starts to converge on the training set. The wide gap between the loss curves on the two validation sets shows that the model overfits the in-distribution split on all tasks, failing to generalize to more difficult samples.

Considering the detailed performance of the model represented in Figure 3.3, we observe that, in the case of Arithmetic and Algebra, the model generalizes better on formulas with more operands than seen during training (data split $(N = 2, O = 4)$), rather than in the case of more deeply nested formulas (data splits $(N = 3, O = 2)$ and $(N = 4, O = 2)$). Since the training set includes examples of a 'base case' and an 'induction step' for both parameters (respectively, $(N = 1, O = 2)$ and $(N = 2, O = 2)$ for Nesting, $(N = 2, O = 2)$ and $(N = 2, O = 3)$ for Operands), this could indicate that it is easier to learn a generalization

step when the complexity of a formula increases in terms of number of operands, rather than in terms of nesting depth.

3.3 Do bigger LLMs reason more systematically?

In this section, we explore the performance of large-scale language models on the tasks described in Section 3.1, focusing mainly on the impact of scaling models. This directly addresses the research questions: Do these models learn to reason systematically, and does this capability improve with scale? Furthermore, we take into account the impact of fine-tuning on the models' systematic reasoning capabilities.

We evaluate models from the Llama family, including the vanilla Llama-2 Chat (or, Instruct) and two variants fine-tuned on mathematical tasks. For all models, we compare three different sizes, allowing us to assess how both scaling and fine-tuning affect reasoning performance.

3.3.1 Models

We evaluated the symbolic reasoning abilities of three models: Llama 2 Chat [126], and two fine-tuned versions, MAmmoTH [127] and MetaMath [128]. For all of them, we considered three model sizes with 7B, 13B, and 70B parameters.

The Llama 2 Chat model is a large language model optimized for dialogue use cases, trained on a mix of publicly available data. It generally performs better than existing open-source models, approaching some of the most powerful closed-source models on a series of safety benchmarks, and it achieves high performance on a variety of tasks ranging from common sense reasoning to world knowledge, reading comprehension, and mathematical problem solving [126].

We also chose to test two recently proposed fine-tuned versions of Llama 2, MetaMath and MAmmoTH, that were designed to improve the mathematical reasoning abilities of the base model using different fine-tuning strategies. MetaMath has been fine-tuned on MetaMathQA, a companion dataset created by bootstrapping samples in the GSM8K and MATH datasets by rephrasing both questions and answers with the aim of increasing variety in the training samples [128]. MAmmoTH was created with the aim of generalizing to many different mathematical and reasoning domains, hence it has been fine-tuned on eight different popular benchmarks and evaluated on both in-domain and out-of-domain problems from different datasets [127].

3.3.2 Data

We benchmark the LLMs in the Llama family on ListOps and the Arithmetic formulas only. We do not include Algebraic formulas in our analysis because we have observed very low performance on these problems which would not allow to draw meaningful conclusions from the comparison of performance on data splits of varying complexity.

We also slightly modified the original format of the ListOps formulas by using a more explicit functional notation since it has been observed that the notation used to represent

symbolic formulas can strongly influence the performance of transformers on arithmetic tasks [129]. For example, the formula `[MAX 3 9 1]` was rewritten in the new format as `MAX(3, 9, 1)`, since this notation is more likely to be observed in other mathematical datasets used for training and fine-tuning of the LLMs considered here. We evaluated models on ListOps formulas drawn from the data split parameterized by $(N = 4, O = 4)$, i.e., that had two to four operands and one to four nesting levels. As regards arithmetic, we considered formulas with one to four nesting levels and two operands, drawn from the split parameterized by $(N = 4, O = 2)$. We do not include formulas with more operands since we observed that the nesting level of the formulas has the most significant impact on performance. Indeed, we present results on ListOps formulas aggregating across the number of operands.

3.3.3 Prompting methods

In order to elicit the emergence of reasoning abilities in Llama 2 Chat, we have initially tested zero-shot chain-of-thought prompting, a recently proposed prompting method that was shown to achieve similar performance as chain-of-thought prompting without the need to craft exemplars [36, 107]. However, we observed that Llama 2 Chat already produced reasoning steps in the output using zero-shot prompting, presumably as the result of fine-tuning with reinforcement learning through human feedback [100], and that zero-shot chain-of-thought prompting did not further improve the model's performance. Therefore, we opted for a simpler zero-shot prompting strategy, in which we briefly describe the task and then directly ask the model to solve it. In the case of the ListOps dataset, we also briefly describe the semantics of the operators that appear in the expression. For example, a Llama 2 prompt to solve a ListOps formula could be the following: *`MIN`, `MAX` and `SM` are operators on lists of single-digit integers which have the semantics of minimum, maximum and sum modulo 10, respectively. Solve the following expression involving these operators: `MAX(3, 9, 1)`. Give the final answer stating 'The final answer is: `<NUMBER>`'.*

In the case of MAmmoTH and MetaMath, we instead used the official prompting strategy used by the authors of the model during fine-tuning. The prompt formats for the two models are similar: they both include an initial sentence introducing a generic task to the model followed by task-dependent instructions and a request to produce a response. In our case, we used the Llama 2 zero-shot prompt as a description of the task to be solved and we inserted it in the official prompt format. The MetaMath prompt contained an explicit request to solve the problem step-by-step, while this was not required with MAmmoTH models since they have been fine-tuned with reasoning problems solved via chain-of-thought prompting, and thus produce a sequence of reasoning steps by default. For example, a MetaMath prompt to solve an arithmetic formula could be the following: *Below is an instruction that describes a task. Write a response that appropriately completes the request. Instruction: Solve the following arithmetic expression: `((86+51)+(-74-35))`. Take the modulo 100 of intermediate values, i.e. keep the last two digits of the number with the sign. Give the final answer stating 'The final answer is: `<NUMBER>`'. Response: Let's think step by step.*

To extract the final answer from the model response we used regular expressions to match integers and take the last one appearing in the text. We measured the performance of all models using sequence accuracy, which means that an output was considered correct only if it exactly matched the target.

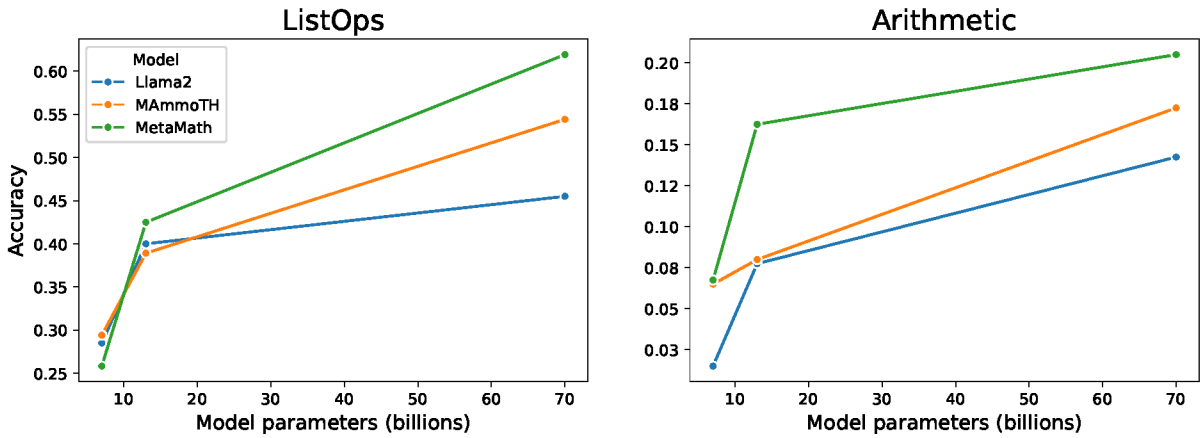


Figure 3.4: Average accuracy on the ListOps and Arithmetic tasks obtained by Llama 2, MAmmoTH, and MetaMath models of increasing size. Larger models (especially if fine-tuned on math tasks) achieve better performance than smaller ones.

3.3.4 Results

In this section, we present the results of the evaluations of the three models. We first focused on the impact of model size on the performance, and then study how performance is modulated by different levels of problem difficulty. Finally, we take a closer look at the errors committed by the models on the simplest examples, to better characterize their capacities and limitations.

Model size improves global accuracy

Fig. 3.4 shows the performance of the three models considered on ListOps and Arithmetic formulas, averaged across formulas of any nesting level and with any number of operands. From this coarse-grained analysis, we can already observe that accuracy always increases with model size. In particular, models with 13B parameters seem to develop significantly better symbolic reasoning abilities than their smaller 7B versions, and accuracy improves further in the largest models.

From the aggregate performance metrics, it is also clear that solving ListOps formulas is much easier than solving arithmetic ones, even for models that have been specifically tuned on mathematical reasoning datasets (note the different scale of the y-axes). This phenomenon could be due to the fact that arithmetic formulas involve complex operations like multiplication between two-digit integers, and calculation of the modulo 100 of intermediate results.

It is also evident that fine-tuning Llama 2 on mathematical problems improves its capacity to solve the kind of math-based symbolic reasoning problems we considered in the experiments. MetaMath models (especially the medium and large versions) outperform MAmmoTH models on both tasks, probably because they have been fine-tuned only on mathematical problems that are more similar to the ones we consider here. Furthermore, in the case of ListOps, we find that fine-tuning is especially effective in boosting the perfor-

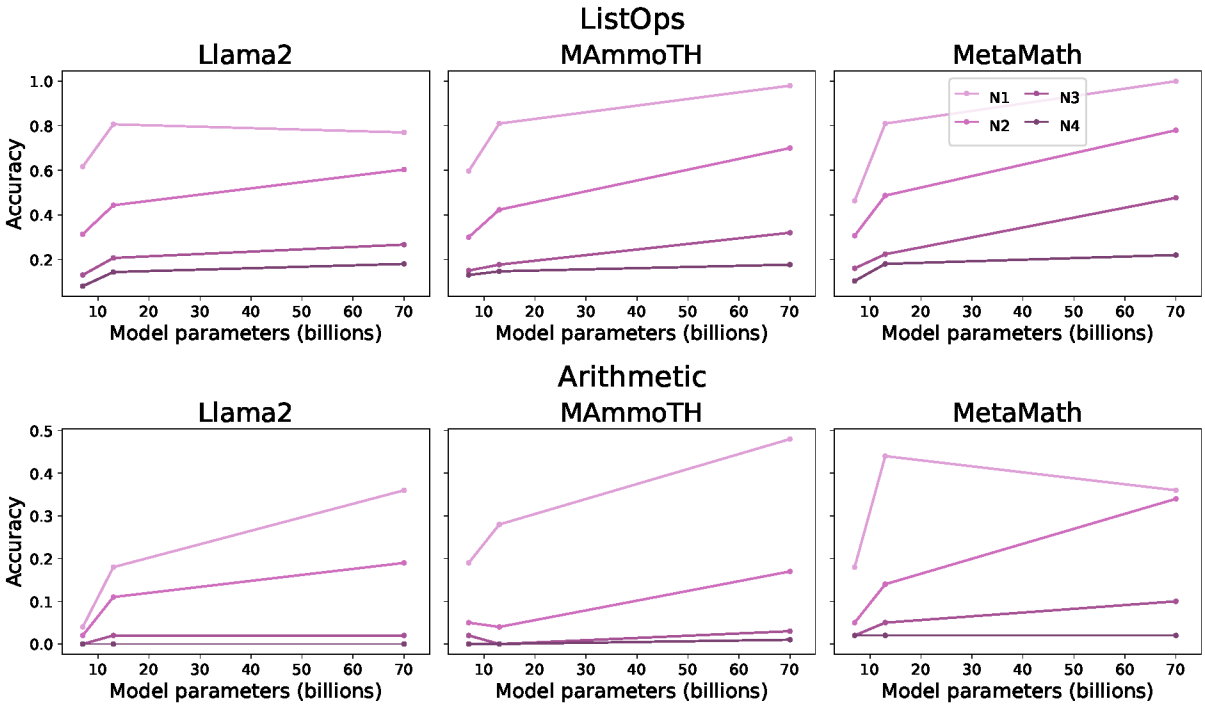


Figure 3.5: Accuracy of Llama 2, MAmmoTH and MetaMath models on ListOps (top) and arithmetic (bottom) formulas of varying levels of difficulty as a function of model size. Nk indicates formulas with nesting level k.

mance of 70B models, leading to gains of 9% (MAmmoTH) and 17% (MetaMath) compared to the 70B Llama 2 model. This could indicate that in the case of tasks consisting of a composition of relatively simple elementary operations, greater performance improvements can be achieved by fine-tuning large models.

Accuracy improves more on simple formulas

We now aim to better understand where the performance gain obtained by larger models is concentrated. In Fig. 3.5, we report a fine-grained measure of the accuracy of the three models on groups of ListOps and arithmetic formulas of increasing nesting levels. As expected, all models are generally more accurate when solving formulas in the easiest splits of both tasks (i.e., formulas with only one or two nested expressions). This indicates that increasing the nesting level of a formula indeed makes the problem more difficult for all the models.

We further notice that as model size grows, accuracy generally increases more on formulas with nesting levels 1 and 2, both for Arithmetic and ListOps. For ListOps we observe a slight improvement in accuracy with model size also for the most complicated formulas, while for the most challenging arithmetic problems the improvement is almost null. It is also interesting to notice that the largest version of MetaMath (70B) is slightly less accurate than the intermediate version (13B) in arithmetic problems with a single nesting, suggesting that scaling-up model size might be detrimental in some cases [130]. As the analyses presented in the following paragraph will show, some light on this result can also be shed considering

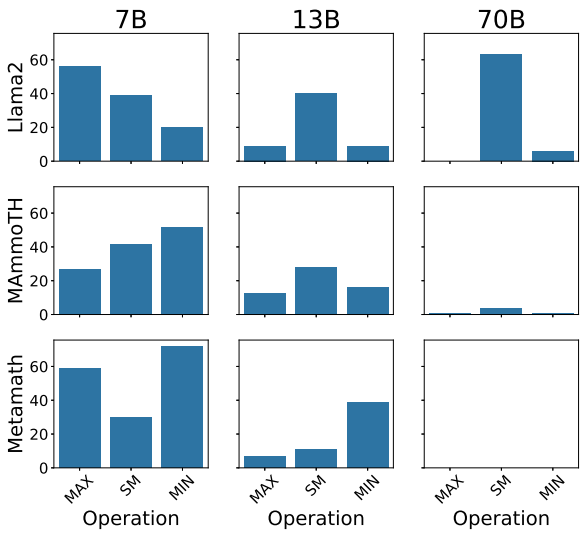


Figure 3.6: Type of errors made by the models on ListOps formulas with a single nesting level. Absolute number of errors is on the y-axes and operator used in the formula is on the x-axis.

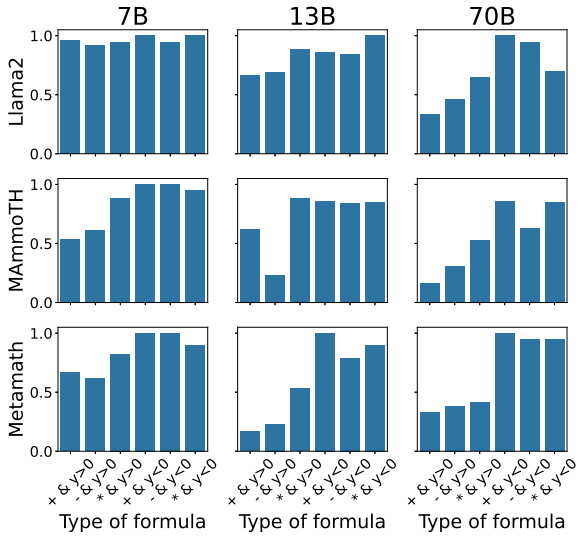


Figure 3.7: Type of errors made by the models on arithmetic formulas with nesting level 1, grouped by operator (+,-,$*$) and sign of the result (y). Incidence of errors is measured by group.

the effect of the modulo operation in simple arithmetic formulas with just one nesting level.

Overall, these results suggest that the emergent symbolic reasoning abilities observed in the largest models do not yet allow for compositional generalization [46], being mostly effective on relatively simple formulas. This holds even for fine-tuned models, since their accuracy on the most challenging problems (four nested formulas) is comparable to that achieved by the base Llama 2 models.

Analysis of errors on simple formulas

In order to solve the symbolic reasoning problems considered in the experiments, the models should be able to solve both atomic operations, such as summing two numbers or finding the maximum in a list, and apply the correct sequence of solution steps in nested formulas. Since we generally observed that performance mostly improved on simpler problems, in the following analysis we focus on studying the capacity of the models to solve the simplest arithmetic and ListOps formulas. To this aim, we isolated and analyzed the errors on data splits with a single nesting level.

In Fig. 3.6, we show the number of errors committed by the models on ListOps formulas with a single nesting. We observe that for Llama 2, the scale of the model allows to significantly improve its ability to solve min and max operations (almost to perfection), while the sum modulo 10 becomes the most difficult operation in the largest model, with a surprising deterioration in performance compared to smaller-scale models. The trend is different for fine-tuned models, which have presumably observed the sum modulo 10 operation more frequently during training and can thus solve it better than the base model, with the largest

model versions reaching almost perfect accuracy on all atomic operations.

In Fig. 3.7, we show the mistakes made by the models on arithmetic formulas with a single nesting level. We group input samples based on the type of operation appearing in the formulas and on the sign of the result, as we hypothesize that operations involving negative operands could be more difficult than those on positive ones. We then measure the incidence of errors in each group, i.e. the fraction of formulas in each group that the model does not solve correctly. We observe that both fine-tuning and reasoning abilities emerging with scale mainly improve the models' accuracy on formulas that have a positive result. By looking at the reasoning steps produced by the models, we noticed that the vast majority of these errors are due to an incorrect calculation of the modulo operation, which indeed proves to be more difficult for all models when it involves negative operands. This could be due to the fact that training samples with modulo operations involving positive integers are significantly more frequent than those involving negative ones, leading the model to develop a bias towards the former.

3.4 Can prompting elicit systematic reasoning?

In this section, we investigate the impact of various prompting methods on the systematic reasoning capabilities of large language models, and specifically on nested formula simplification. This exploration addresses the research question: How reliably can systematic reasoning be elicited through prompting techniques?

Given the positive impact of scale observed in previous analyses, we now focus on very large-scale models – indeed, even much larger than models in the Llama 2 family. We thus benchmark GPT-3.5 and GPT-4, comparing multiple prompting methods—both general-purpose ones and methods recently proposed specifically for reasoning tasks. We aim to determine whether prompting methods can enable the models to exhibit systematic, algorithmic behavior and to what extent they improve performance on structured tasks of varying difficulty.

3.4.1 Models and data

We benchmarked two large language models in the GPT family, namely GPT-3.5 and GPT-4 [131]. We prompt the models to solve formulas drawn from the same nine data splits used to test the Neural Data Router, namely the ones parameterized by values in the Cartesian product of the sets $N = \{2, 3, 4\}$ and $O = \{2, 3, 4\}$.

3.4.2 Prompting methods

In the following, we present the prompting techniques involved in this study. Some of these methods have been mentioned in Section 2.4 Examples of each prompting technique on the three tasks are reported in paragraph A.2 in the Appendix.

Zero-shot

This prompting technique simply consists in giving the problem description as input to the model and directly asking for the result. The question is formatted in such a way that the output of the model will be constrained to the desired format and can thus be easily parsed.

Few-shot

One of the abilities that had a big impact on the popularization of LLMs is their capacity to learn from examples at inference time, a technique called 'few-shot' or 'in-context' learning [13]. In this case, we prompt the models by providing a list of three examples of solved formulas before asking to solve the actual problem. The formulas are sampled from three data splits described by the following values of the Nesting and Operands parameters: $(N = 1, O = 2)$, $(N = 2, O = 2)$ and $(N = 2, O = 3)$. In the examples, the formulas are solved directly, i.e. without showing intermediate solution steps to the model.

Chain-of-Thought

One of the most general and effective techniques that have been recently proposed to elicit reasoning in Large Language Models is Chain-of-Thought (CoT) prompting [106]. When prompted following this method, the model receives a set of examples showcasing the solution of a given reasoning problem, where each example explicitly includes the intermediate solution steps required to get to the final answer.

In our case, we experiment with two different kinds of CoT prompting: in the first one, named 'Symbolic Chain-of-Thought', we provide the solution examples to the model exclusively in a symbolic form, that is, as a chain of equalities. In the second case, named 'Verbal Chain-of-Thought', each intermediate step is also described with English text, suggesting the model a motivation for taking that simplification step and encouraging it to mimic the same verbalization behavior when producing the answer to the actual problem.

Role assignment

We also experiment with a variant of the Zero-shot prompting technique in which we assign a role to the agent. Following recent findings [132] which suggest that specifying the agent's field of expertise could improve the accuracy of its answers, we input the sentence "You are a brilliant mathematician" before giving the model the actual problem description.

Zero-shot Chain-of-Thought

Zero-shot Chain-of-Thought prompting [107] is a technique which has been proposed to obtain similar results as the ones obtained with Chain-of-Thought prompting, without the need to carefully engineer prompts with examples demonstrating the solution steps. The model is prompted directly with the problem it needs to solve, as well as with the first words of the answer: "Let's think step-by-step". The output generated by the model is then collected and used to prompt the model a second time to get the final answer in the correct format, now eliciting the output with the usual formula: "So, the final answer is:".

	ListOps	Arithmetic	Algebra
GPT-3.5			
Zero-shot CoT	0.44	0.32	0.19
Self consistency	0.56	0.35	0.26
GPT-4			
Zero-shot	0.33	0.04	0.10
Zero-shot role	0.42	0.06	0.18
Few-shot	0.46	0.08	0.19
Symbolic CoT	0.48	0.14	0.24
Verbal CoT	0.58	0.29	0.25
Zero-shot CoT	0.71	0.49	0.39
Self-consistency	**0.79**	**0.58**	**0.52**

Table 3.2: Average performance of GPT-3.5 and GPT-4 with different prompting methods on both in-distribution and out-of-distribution test splits, measured in terms of percentage accuracy. "Zero-shot role" refers to the Zero-shot prompting method where the agent was assigned a role. The best performance for each task is highlighted in bold.

Self-consistency

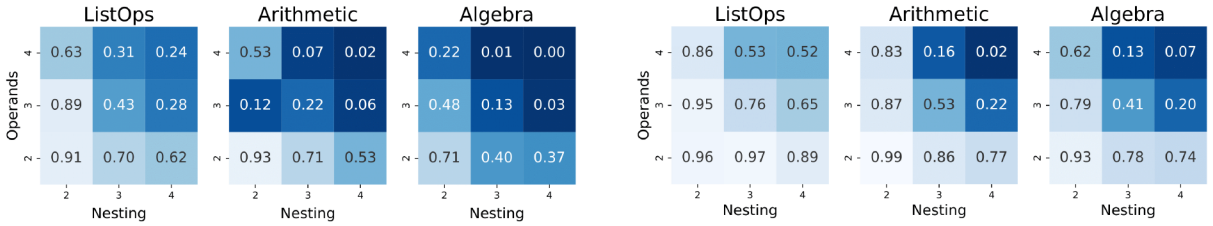
Reasoning problems are different from other problems that can be tackled with generative models, in that they always have a unique solution (at least semantically, i.e. not taking into account the different ways in which a solution can be written, for example in the case of algebraic expressions). Nevertheless, there could be multiple reasoning paths that lead to the correct solution, varying not only formally, but also substantially – as in, for example, theorem proving. Starting from this premise, [109] advocate for *self-consistency* in Large Language Models' outputs when solving reasoning tasks. The basic idea is that the performance of the model might improve if we prompt the model several times, and consider the answer that was generated more frequently and therefore, one might say, with more confidence.

We apply the self-consistency prompting method in combination with Zero-shot CoT prompting. To limit the consumption of credits to query the OpenAI API, we prompted the model only 5 times for each input, rather than 40 times as done in the original work. We note, therefore, that the models performance might further improve raising the number of prompts per input, and thus the confidence in the selected answer. However, even with such a small number of outputs, we can already observe the effectiveness of this prompting technique.

3.4.3 Results

Prompting methods and tasks

As reported in Table 3.2, on all tasks the best performance was achieved by GPT-4 using the Self-consistency prompting method. More generally, prompting techniques that require (or encourage) GPT-4 to reason explicitly were more effective: in all cases, the best perfor-



(a) GPT-3.5 (b) GPT-4

Figure 3.8: Performance of GPT-3.5 and GPT-4 using Self-consistency prompting on the test splits. Values represent output accuracy in percentage: the performance of all models and prompting methods clearly decreases on data splits of higher complexity.

mance obtained by prompting methods that ask the model to directly provide the answer (either with or without solution examples) is significantly lower than that achieved by Self-consistency prompting. At the same time, it is interesting to note that by just contextualizing the role of the agent (Zero-shot role prompting) we can improve the model's accuracy in all tasks, compared to Zero-shot prompting.

Comparing Chain-of-Though prompts with symbolic solution steps and verbalized solutions steps, we found that asking the model to spell-out the reasoning process significantly improved the performance on the ListOps and Arithmetic tasks, while it had limited impact on the Algebra task. Furthermore, we can see that the task for which CoT prompting is more useful is clearly Arithmetic, with a gain in accuracy up to +40% compared to Few-shot prompting. While it is hard to single out a single reason why a given prompting method could be more effective in one particular task, we conjecture that Arithmetic could particularly benefit from the explicit formulation of solution steps due to the considerable difficulty of computing the product between two-digits integers, which for a Large Language Model is probably the hardest elementary operation present in the datasets.

Prompting methods and generalization

Taking into account the performance of GPT-4 on the nine different data splits considered for each task, we can make some considerations on the capacity of different prompting methods to enable systematic generalization.

We show in Figure 3.9 the performance gain from each prompting method compared to the Zero-shot baseline, measured in terms of difference in percentage accuracy. We observe that, in general, the performance gains resulting from the prompting techniques that lead the model to make explicit reasoning steps are concentrated in the data splits with low levels of complexity. The performance gain on data splits parameterized with $(N = 3, O = 4)$, $(N = 4, O = 3)$ and $(N = 4, O = 4)$ is generally less evident.

As previously noted, this general tendency is more evident on Arithmetic, where the performance gains deriving from prompting methods that produce explicit reasoning are greater overall. On the other hand, the performance gain from such prompting methods is more evenly spread across data splits in the case of ListOps, for which the GPT-4 performance on simple splits is already quite high even using Zero-shot prompting, as shown in

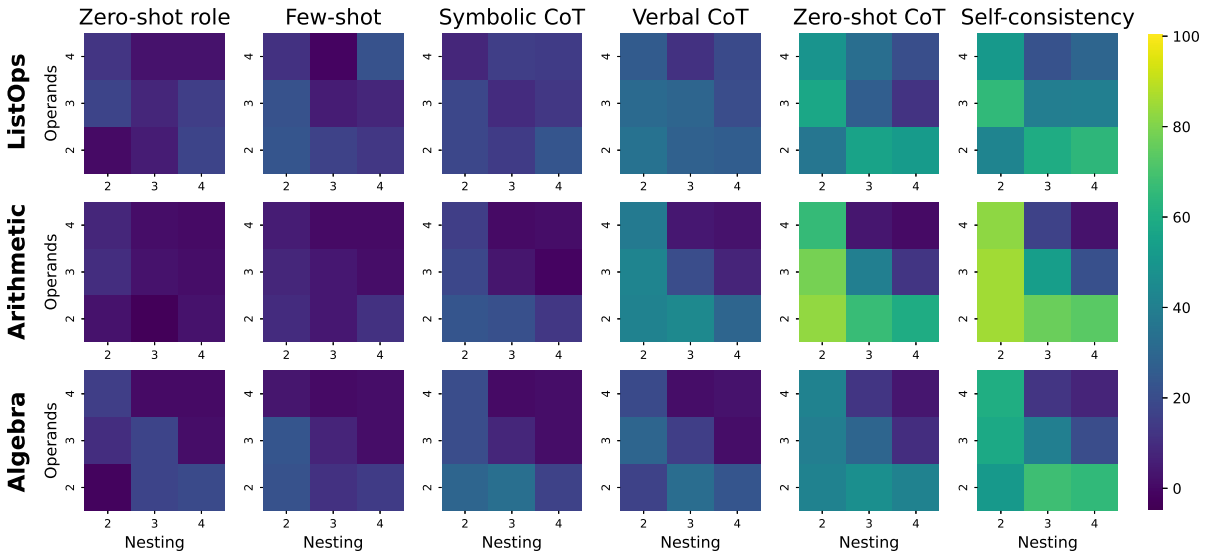


Figure 3.9: Performance gain measured as percentage accuracy resulting from each prompting method on GPT-4 compared to Zero-shot baseline. The accuracy gains from the best prompting methods are concentrated in simpler data splits, especially on Arithmetic.

Table 3.2 (see also paragraph A.1 in the Appendix).

Overall, this suggests that prompting methods in the Chain-of-Thought family, by letting the model produce explicit reasoning steps, do improve the performance of GPT-4 on the problems at hand to some degree on simple problem instances, but they do not trigger the emergence of a reasoning mechanism that can lead to systematic generalization that is independent of the problem difficulty.

3.5 Conclusions

In this chapter, we investigated the systematic reasoning capabilities of both small-scale models and large-scale language models (LLMs) on a novel formula simplification task, designed to reflect core aspects of compositionality as outlined by Hupkes et al. [46]. The task allowed for an analysis of model performance across varying levels of formula complexity, enabling a detailed exploration of the key research questions we outlined previously.

The results demonstrate that the Neural Data Router, which we take as a representative of small-scale neural architectures specialized on systematic reasoning tasks, performs well on simpler instances but faces significant challenges as task complexity increases. This points to a bottleneck in its ability to handle increasing complexity, suggesting that specialized architectures may struggle to extrapolate on more complex tasks than the ones for which they were designed.

The findings for LLMs present a more nuanced picture. Scale proves to have an impact on performance: larger models generally outperform smaller ones in solving mathematical formulas, especially for simpler cases. However, the gains are not uniform across all task complexities. For more deeply nested formulas and tasks involving higher degrees of compo-

sitionality, LLMs still exhibit gaps in systematic reasoning. These results suggest that while LLMs demonstrate emergent symbolic reasoning at larger scales, the capacity to apply this reasoning reliably to complex, multi-layered structures remains incomplete.

Prompting techniques have shown some potential to elicit systematic reasoning in LLMs, particularly when prompts explicitly articulate intermediate reasoning steps. However, this method did not lead to consistent generalization across all tasks, especially when the complexity of the formulas increased. While prompting can guide LLMs toward more structured reasoning, it does not fully close the performance gap on more challenging problems. This highlights that although prompts can improve the elicitation of reasoning, they are not a comprehensive solution for ensuring systematicity, especially at higher levels of compositional complexity.

Overall, these findings align with broader research in the field, suggesting that, although progress has been made, significant challenges remain in achieving reliable systematic generalization in both small- and large-scale neural networks.

Chapter 4

Learning neuro-symbolic convergent term rewriting systems

A fundamental challenge in artificial intelligence is designing systems that can learn to reason effectively. By 'learning to reason,' we refer to the ability to apply learned information processing mechanisms to unseen and potentially more complex inputs, especially of structured kind, in a systematic and reliable way. However, achieving this requires addressing several core difficulties in how current systems, particularly those based on neural networks, learn and process information. As we have seen in the previous chapter, despite a consistent amount of research on systematic and compositional generalization in both small-scale and large-scale neural networks, both kinds of systems struggle to generalize learned reasoning patterns to out-of-distribution samples or highly complex problems. This limitation severely hampers their ability to tackle structured tasks like those commonly solved with classical algorithms. In this chapter, we consider this issue focusing on a specific class of reasoning problems that can be formalized as convergent term rewriting systems [133]. We propose a neuro-symbolic framework that can be implemented in systems with strong generalization capabilities on the class of problems at hand. These capabilities are achieved by leveraging both background knowledge on the properties of the problems and their ideal solution process, and out-of-distribution capabilities of transformers with key architectural modifications.

4.1 Introduction

Classical algorithms in computer science – such as sorting or graph traversal algorithms – can map inputs to outputs independently of the data distribution from which the input was drawn, with well-studied time and space scaling laws. On the other hand, from a statistical perspective inputs to deep learning models are almost always assumed to belong to the same data distribution of training samples, making it challenging to design learning architectures that can handle out-of-distribution (OOD) test data [134, 135]. At the same time, however, the design and implementation of classical algorithms requires a significant amount of human labor: the programmer needs to formalize each problem class using specific data structures and engineer explicit algorithms that can solve the problems at hand. Given the immense

wealth of digital data that is now available to organizations and the value they can bring if processed by computer programs, new lines of research propose to reduce the need for humans in the automation loop exploiting machine learning. In this context, information is represented in vectors and is processed by manipulating these representations rather than with classical algorithms and data structures [42]. For example, graph neural networks have recently been used to learn graph algorithms, combining deep learning's strong real-world data handling capabilities with classical algorithms' theoretical guarantees [90, 136].

Here we consider a class of problems from the tradition of artificial intelligence and computer science that can be formalized as convergent term rewriting systems [133]. Generally speaking, rewriting systems are composed of a set of elements and a set of rules that describe how to transform those elements. Elements can be several different mathematical objects, including strings, graphs or terms of a formula. When combined with an appropriate algorithm, rewriting systems become programs that can execute the transformation of a sequence of objects into another one by the subsequent application of the given rules. We consider here term rewriting systems in which the elements are mathematical expressions represented as sequences of symbols, and the rules define their semantically equivalent forms. Specifically, in a *convergent* term rewriting system, rewrite rules applied sequentially always transform the input into the same final output, independently of the order of application, and sequences of rewrite rules never form loops.

In this chapter, we aim to address several related research questions about the design and implementation of a neuro-symbolic system capable to learn the problems described above. The research questions we address are the following: How can a hybrid neuro-symbolic architecture be designed specifically for the class of problems we consider? What is the impact of different design elements on the system's performance? And, finally, what is the impact of these design choices on efficiency, measured in terms of training time, inference time, and memory consumption?

To address the first question, we present here in detail two related neuro-symbolic architectures designed to learn convergent term rewriting systems: the Neural Rewriting System (NRS) and the Fast Neural Rewriting System (FastNRS). Both models are built upon a shared architectural blueprint inspired by rewriting algorithms. They both exhibit strong generalization capabilities, akin to those of traditional term rewriting systems, but their processing dynamics emerges through learning from data rather than manual design. The models can be trained on a limited subset of formulas and effectively generalize to more complex ones, eliminating the need for exhaustive training on all possible formulas. Such capability for out-of-distribution generalization is enabled by a modular approach informed by the rewrite mechanism and by architectural modifications to the transformer block.

Additionally, we show that both models can function in a multi-domain scenario, a setting where a single model is trained on multiple datasets simultaneously without task-specific architectural adjustments. This results in a "multi-potent" system capable of solving a variety of problem instances within the considered class. Unlike the more conventional multi-task setting in machine learning [137], where a shared backbone architecture is typically combined with task-specific outputs, our architecture's algorithmic-inspired design allows the same components to be effectively applied to learn multiple term rewriting systems, enabling robust generalization without task-specific modifications.

To address the second two research questions and understand the impact of model de-

sign choices on the functioning of the models, we provide a comparative analysis of their performance and efficiency on the four different domains of logic, lists, integer arithmetic, and simple algebra, describing the benefits and trade-offs associated with each system.

As additional baselines that could exhibit a systematic behavior similar to the execution of convergent term rewriting systems, we consider three architectures in two independent but related streams of research: the Neural Data Router [70], a recently proposed variant of transformer designed to achieve strong systematic generalization capabilities, as a representative of small-scale neural architectures specialized on single tasks; OpenAI's GPT-4 [138], one of the best performing general-purpose LLMs currently available, whose reasoning capabilities have been recently studied and improved via specialized prompting methods like Chain-of-Thought prompting [106]; and, OpenAI's o1-preview [139], a recently proposed large-language model based on GPT-4 and designed to significantly improve performance on complex reasoning tasks by optimizing the production of very long reasoning chains.

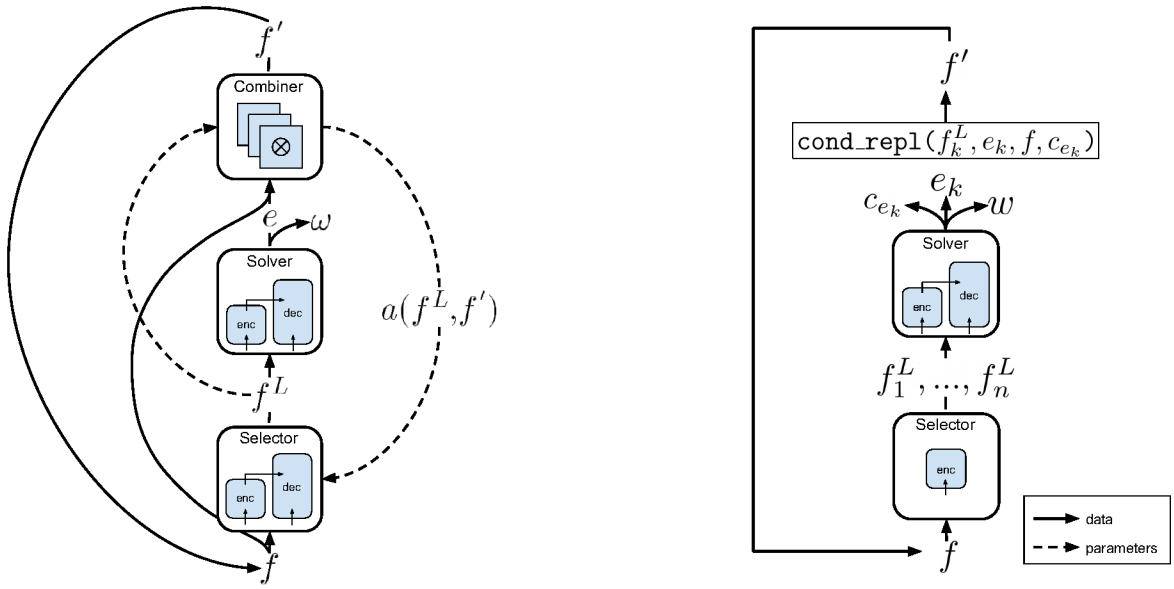
4.2 Formula simplification problems: a formal description

Convergent term rewriting systems are typically used to simplify mathematical formulas. Here, rules describe how expressions involving two or more operands can be rewritten into atomic elements that are semantically equivalent and represent their values. For example, in arithmetic formulas, the expression $(15 + 5)$ can be transformed into the equivalent atomic element 20. We call these problems "formula simplification problems." We will now formally characterize these problems and then use this formalization to describe how convergent term rewriting systems for this class of problems can be formed, i.e., how rewriting rules can be written and what algorithm is implemented by such a rewriting system.

In formula simplification problems, we can see formulas $f \in F$ as entities that are composed of two semantically distinct elements: operators $o \in O$ and arguments $a \in A$. Arguments can be either atomic elements $e \in E$, such as integers, which are also the final values of any formula, or other formulas. Furthermore, since these problems can be solved by convergent term rewriting systems and thus always have exactly one final value, the following equality holds for any formula: $f = o(a_1, ..., a_n) = e$, where $o \in O$, $e \in E$ and $a_j \in F \cup E,\ \forall j \in [1, n]$. Finally, we can define leaf formulas $F^L \subset F$ as the subset of formulas whose arguments are all atomic elements: $f^L = o(a_1, ..., a_n)$ s.t. $a_k \in E,\ \forall k \in [1, n]$.

For any problem, we can define a set of rewriting rules $r \in R$ which map leaf formulas to their values: $r : F^L \to E,\ \forall r \in R$. This set defines a convergent term rewriting system for any formula simplification problem. Indeed, iteratively applying the rewriting rules described above in any order, the initial formula can be simplified to an atomic final element e. The algorithm implemented by the rewriting system thus consists of the execution of four steps, which are repeated until the formula becomes an atomic value: **1.** pick any valid rewriting rule from the set;[1] **2.** find in the input formula all the elements that match the left-hand side of the rewriting rule; **3.** apply the rewriting rule to the elements found and compute the substitution; **4.** replace the elements with the computed values. This algorithm serves as a

[1]In step **1.**, a rewriting rule that can be applied to the current formula should be chosen, i.e., one whose left-hand side appears in the formula.



(a) NRS architecture. The three modules in the NRS operate sequentially, but the Selector and the Combiner also interact via the agreement score $a(f^L, f')$ to produce the Selector output, as described in Paragraph 4.3.1.

(b) FastNRS architecture. The Selector module in the FastNRS operates by selecting n leaf formulas in parallel with a text-segmentation mechanism, as described in Paragraph 4.3.2. Each leaf formula is then processed independently by the Solver.

Figure 4.1: Schematic representations of the NRS and FastNRS architectures. Both architectures implement the three modules of the algorithmic blueprint described in Section 4.3. The models process input formulas f, selecting one or more leaf formulas f^L. Solver modules simplify leaf formulas to atomic values e and produce a special end-of-computation token w. Combiner modules produce simplified formulas f'.

blueprint for the design of our architectures, providing strong guarantees on the reliability of the models, regardless of how they are implemented.

4.3 Neuro-symbolic architectures

Traditional term rewriting systems rely heavily on manually crafted rules and algorithms tailored to specific problems, requiring significant human expertise and effort. In contrast, our approach aims to *learn* both the rewriting rules and the contexts in which these rules should be applied. By mirroring the algorithm's steps in the architecture design itself, we adopt a structured approach similar to that of classical algorithms. At the same time, combining this with the flexibility of learning-based methods, we obtain a versatile system that can be applied to different problems within the class we consider. This adaptability is particularly advantageous in a multi-domain scenario, where the same architecture can be effectively employed for diverse problems, as we demonstrate with experimental results.

Compared to other neural systems, such as general-purpose Large Language Models

(LLMs) and specialized neural architectures for compositional reasoning, our system balances generality and reliability. Indeed, while LLMs can tackle a wide range of tasks, they lack the specialized focus needed to provide strong guarantees in solving structured problems like those that can be addressed by term rewriting. On the other hand, small-scale transformer-based architectures designed for compositional reasoning tend to provide stronger guarantees of generalization, but the spectrum of tasks they can handle is limited, if at all defined [117]. Our neuro-symbolic system is more specialized than LLMs yet more versatile than narrowly specialized architectures, offering a structured and more reliable solution for solving formula simplification problems across different domains.

The algorithm for resolving mathematical formulas we described in Section 4.2 can be implemented in a neuro-symbolic architecture in various ways, depending on design choices and priorities. We present here two neuro-symbolic implementations of this algorithmic blueprint: the Neural Rewriting System (NRS) and the Fast Neural Rewriting System (Fast-NRS). Both these architectures are composed of three modules, the Selector, the Solver and the Combiner, which handle steps **1.** and **2.**, **3.** and **4.** of the algorithm, respectively. Note that in a neural implementation, picking a valid rule can coincide with identifying a part of the input that can be simplified, as it will be clear from our implementation of the Solver. These designs, represented schematically in Figures 4.1a and 4.1b, primarily differ in how the Selector module is implemented, corresponding to steps **1.** and **2.** of the algorithm—the identification of matching elements for rule application. In the Neural Rewriting System, only one element matching the left-hand side of a rewriting rule is selected at a time. On the other hand, the Fast Neural Rewriting System selects and replaces multiple elements in parallel by framing the problem as a text segmentation task, allowing for a more efficient implementation at the expense of some accuracy.

Algorithm 1 Pseudo-code of the NRS execution. The model executes the Selector, Solver and Combiner in a pipeline. Algorithm 2 describes the execution of the Selector.

```
function NRS(f)
    while True do
        f^L, c(f^L), a(f^L, f) ← SELECTOR(f)
        if a(f^L, f) ≠ 1 then
            return ∅
        end if
        e ← SOLVER(f^L)
        if e = ω then
            return f
        end if
        f ← COMBINER(f, f^L, e)
    end while
end function
```

4.3.1 The Neural Rewriting System (NRS)

The general functioning of the architecture can be described as the iterative execution of the Selector, Solver and Combiner modules in a pipeline. A formal description in pseudo-code of the NRS execution is given in Algorithms 1 and 2.

Algorithm 2 Pseudo-code of the NRS Selector execution. The TRFM function represents the transformer. The model conditionally applies Dynamic Windowing (lines 9-13) depending on the input length.

Require: parameters M, T

1: **function** SELECTOR(f)
2: $L \leftarrow [\,]$
3: **for** $i = 1 \rightarrow M$ **do**
4: **if** $|f| < T$ **then**
5: $\hat{f}^L, c(\hat{f}^L) \leftarrow \text{TRFM}(f)$
6: $a(\hat{f}^L, f) \leftarrow \frac{\max\ CNN_{\hat{f}^L}(f)}{|\hat{f}^L|}$
7: Append $\langle \hat{f}^L, c(\hat{f}^L), a(\hat{f}^L, f) \rangle$ to L
8: **else**
9: $k \leftarrow \text{floor}(|f| \cdot \frac{i \bmod 20}{20})$
10: $f_w \leftarrow w(f, k)$
11: $\hat{f}^L, c(\hat{f}^L) \leftarrow \text{TRFM}(f_w)$
12: $a(\hat{f}^L, f_w) \leftarrow \frac{\max\ CNN_{\hat{f}^L}(f_w)}{|\hat{f}^L|}$
13: Append $\langle \hat{f}^L, c(\hat{f}^L), a(\hat{f}^L, f_w) \rangle$ to L
14: **end if**
15: **end for**
16: Sort L by $a(\hat{f}^L, f)$ and $c(\hat{f}^L)$
17: **return** $L[0]$
18: **end function**

The Selector module

The Selector module in the Neural Rewriting System is responsible for identifying an element in the input formula that matches the left-hand side of a rewriting rule. As mentioned earlier, it is designed to solve a sequence-to-sequence task. Formally, it implements the $sel : F \rightarrow F^L$ function, i.e. it is trained to map a formula, which we assume to always be syntactically correct, to a leaf formula appearing therein. In analogy to what happens in humans when they deploy object-based attention to locate algebraic sub-expressions that can be simplified [140], the Selector is trained to identify the last leaf formula occurring in the input formula on which a rewriting rule can be applied. For example, given the arithmetic expression `(12+(3-(4+5)))` the Selector's task is to correctly identify the solvable leaf formula `(4+5)`.

We use a variant of the transformer encoder-decoder [52] to implement the core of the NRS Selector. In order to achieve strong length generalization capabilities, we make two modifications to the vanilla transformer. First, we follow recent evidence showing that

length generalization in transformers can be influenced by the choice of positional encodings, especially when, at test time, these fall out of the range observed during training [58, 61, 141]. We thus use Label-based Positional Encodings [63] to enable the Selector to identify leaf formulas in very long sequences. The positional information of an input sequence of L tokens is thus encoded in the following way: given a sequence of N sinusoidal positional encodings, where N is a large number that represents the maximum expected length of an input, L integers are sampled in the interval $[0, N-1]$ and then sorted. The encodings found in the positions corresponding to the sampled integers are then summed to the embeddings of the tokens in the input sequence before the forward pass. Notice that the sampling and sorting mechanisms are applied internally in the transformer, similar to the pooling operations in convolutional networks. Furthermore, this type of positional encoding operates as a sort of data augmentation mechanism and thus is not learned and is not involved in the backpropagation of the errors.

As a second modification, we constrain the receptive field of the self-attention layer of the encoder, inspired by sparse self-attention patterns proposed in the literature [55]. This choice can also be intuitively motivated for our tasks. Indeed, the Selector more likely learns a function with good length generalization properties in a smaller search space that contains functions with minimal dependencies on parts of the sequence that do not correspond to leaf formulas. Furthermore, identifying leaf formulas is a local problem in any part of the input sequence, i.e., it can be solved without integrating information carried by tokens located in distant parts of the input sequence. Therefore, we mask all entries in the self-attention matrix of the encoder but the ones around the main diagonal (i.e., we make them $-\text{inf}$). The active values in the self-attention matrix are thus located in a diagonal window that is $2k+1$ tokens wide, where k is a hyperparameter. Preliminary experiments showed that models with a vanilla self-attention achieved worse out-of-distribution generalization, and hyperparameter selection demonstrated that the strongest generalization can be achieved with a narrow diagonal window.

Other than these architectural modifications, the NRS Selector includes two specialized mechanisms – the multi-output generation and the dynamic windowing – that enhance accuracy and reliability, ensuring higher resilience to noise and errors in the neural network's outputs. The multi-output generation mechanism was introduced after we observed experimentally that it can be useful to repeat the auto-regressive generation of transformer outputs. After sampling several output sequences from the probability distribution derived from the decoder's outputs, we choose the best one considering both a measure of confidence of the Selector and a measure of input-output agreement computed by the Combiner.

Given the specialized purpose of the Selector, we can see each output of the module as a candidate leaf formula $\hat{f}^L$. We generate any token $\hat{f}^L_i$ in an output sequence $\hat{f}^L$ by sampling from the probability distribution obtained applying the softmax function to the logits produced by the final fully-connected layer of the decoder. We do not use any temperature parameter when sampling the output tokens. For any input formula f, we repeat the stochastic generation process M times, thus generating a sequence of candidate leaf formulas $\hat{F}^L = \langle \hat{f}^{L,1}, ..., \hat{f}^{L,M} \rangle$.s We define the confidence of the Selector on any $\hat{f}^{L,j}, 1 \leq j \leq M$ as the joint probability of sampling its tokens: $c(\hat{f}^{L,j}) = \prod_{i=1}^{N} p_i^j$, where N is the number of tokens in $\hat{f}^{L,j}$, and p_i^j is the probability to sample token $\hat{f}^L_i$ in $\hat{f}^{L,j}$. We also define an

agreement score $a(\hat{f}^{L,j}, f) \in [0,1]$ which gives information about the fraction of $\hat{f}^{L,j}$ which is exactly present in the input formula f. This measure is computed by the Combiner and thus it is formally defined in Section 4.3.1. We then select the final output f^L of the Selector as the one with the highest confidence which has an agreement score equal to 1 — that is, it matches the input sequence exactly. More formally, $f^L = \hat{f}^{L,j} \in \hat{F}^L$ s.t. $c(\hat{f}^{L,j}) \geq c(\hat{f}^{L,k})\ \forall j,k \in [1,M] \wedge a(\hat{f}^{L,j}, f) = 1$.

We also implement a dynamic windowing mechanism on longer input sequences that allows us to increase the model's generalization capacity on complex problem instances. The core idea behind this mechanism is to repeat the process of selecting a leaf formula several times, changing each time the *window* of the input formula that the Selector observes, and then relying on the confidence $c(\hat{f}^L)$ to pick the best output. We apply this mechanism on top of multi-output generation by modifying its behavior for sequences longer than a given threshold T. Given an input formula f, if $|f| < T$ the computation is executed as described before. Otherwise, we generate M copies of the input $\langle f^{(1)}, ..., f^{(M)} \rangle$, whose lengths will be reduced by applying a window function w. Considering any input f as a sequence $f_1, ..., f_N$ of N tokens, we define the window function $w(f,k) = f_{k+1}, ..., f_N$ which reduces the length of the input by giving as output its last k tokens. Since the Selector is trained to output the last leaf formula appearing in the input, the window function reduces the input length starting from the first tokens. We divide the sequence of copies of the input $\langle f^{(1)}, ..., f^{(M)} \rangle$ into 20 groups $F^{(1)}, ..., F^{(20)}$ of equal size. Intuitively, in each group the length of the input is reduced by a different percentage of tokens. More formally, the window function will be parameterized by $k = \text{floor}(|f^{(i)}| \cdot \frac{j}{20})\ \forall f^{(i)} \in F^{(j)},\ \forall j \in [1,20]$. We then pick the final leaf formula using the confidence and agreement scores, as described in the previous paragraph. This ensures that the model can observe the whole input sequence and select a leaf expression in the part of the input where it can identify one with more confidence.

The Solver module

The Solver is a central component of our system. Indeed, as we described in Section 4.1, classic rewriting systems are composed of a set of elements and a set of rules, which are then used in the algorithm to transform sequences. In our neuro-symbolic architecture, both elements and rewriting rules are represented sub-symbolically in the Solver, which rewrites relevant parts of the input.

Given a leaf formula f^L by the Selector, the Solver is trained to produce the equivalent reduction e according to the corresponding rewriting rule. Therefore, valid elements and rewriting rules are stored implicitly in the network weights via optimization. For example, given the leaf formula `(4+5)`, the Selector produces its value, `9`. The Solver also learns to recognize the termination state of the computation, signaling when such a state is reached. Given atomic elements representing the final value of a formula, such as the number `9` for an arithmetic formula, it is trained to map them to the special symbol ω, indicating the end of computation. During training, the Solver only observes well-formed leaf formulas and atomic values.

We frame the Solver task as a sequence-to-sequence problem. We implement it as a transformer encoder-decoder without any modification since it learns input-output mappings corresponding to the rules.

The Combiner module

The last module in the architecture is the Combiner, a neural implementation of the function $com : F \times F^L \times E \to F$. Its purpose is thus to produce a simplified version of the original formula, given the formula itself f, the leaf formula f^L identified by the Selector, and its reduction e computed by the Solver.

In order to carry out its task, the first operation that the Combiner must perform is finding the position in f where the leaf formula f^L appears. We notice that the convolution is a suitable operation to detect which portion of an input sequence has the highest match with another sequence used as a filter, so we implement this operation using a 2D Convolutional Neural Network (CNN) whose filters are set dynamically at execution time using the output of the Selector, rather than being learned with backpropagation. For example, if we have the arithmetic expression `(12+(3-(4+5)))` as input, using the leaf formula `(4+5)` as the filter of a 2D CNN we can obtain a signal of correspondence between the leaf formula and the input expression, and therefore identify if the leaf formula is present in the input and where it is located.

More precisely, we represent both the input sequence f and the leaf formula f^L as sequences of 1-hot vectors over the same vocabulary. Since the leaf formulas found for different sequences in a batch can have different lengths, we pad each one with zeros to prevent the padding to match in the input. Then, we set the filter of the 2D CNN to the 1-hot representation of f^L. We refer to the CNN parameterized in this way as CNN_{f^L}. Doing so allows us to obtain from the output of the convolution both information on the location of the best match of f^L in f and on the number of tokens in f^L that match f exactly in some point. Indeed, we can compute the location of the best match as $\text{pos}(f^L, f) = \text{argmax } \text{CNN}_{f^L}(f)$. Furthermore, we can compute the agreement score $a(f^L, f) = \frac{\max \text{CNN}_{f^L}(f)}{|f^L|}$, where $|f^L|$ is the number of tokens in f^L. Dividing by $|f^L|$ makes the score normalized, which allows us to compare the agreement scores of leaf formulas with different lengths. Indeed, as described in Section 4.3.1, the Selector uses this score for multi-output generation to discard the outputs that do not have an exact match in the input formula. Notice that in this case the CNN is parameterized using candidate leaf formulas $\hat{f}_j^L$ whose accuracy scores with f are compared. If there is no Selector output such that $a(\hat{f}_j^L, f) = 1$, the computation on the input sequence f is stopped, and this is considered a failure of the model.

After finding the position of the leaf expression in f, the Combiner replaces f^L with e in f, to compute the simplified version of the formula f'. We implement this operation as a deterministic operator with input f, f^L, e, and $\text{pos}(f^L, f)$.

4.3.2 The Fast Neural Rewriting System (FastNRS)

The general functioning of the architecture can be described as the iterative execution of the Selector, the Solver and a deterministic `cond_repl` function in a pipeline. A pseudo-code description of the FastNRS execution is shown in Algorithm 3.

Algorithm 3 Pseudo-code of the FastNRS execution. The model iterates through the leaf formulas extracted by the Selector and conditionally replaces them in the main formula.

function FastNRS(f)
  while True **do**
    mask $\leftarrow$ Selector(f)
    $\langle f_1^L, ..., f_n^L \rangle \leftarrow$ Extract(mask, f)
    replaced $\leftarrow$ False
    for f_k^L in $\langle f_1^L, ..., f_n^L \rangle$ **do**
      $e_k, c_{e_k} \leftarrow$ Solver(f_k^L)
      if $e_k = \omega$ **then**
        return f
      end if
      f, replaced $\leftarrow$ `cond_repl(`f_k^L, e_k, f, c_{e_k}`)`
    end for
    if replaced = False **then**
      return $\varnothing$
    end if
  end while
end function

The Selector module

Unlike the NRS, the FastNRS Selector is implemented using only a transformer encoder. This module shares the same core architecture as the transformer encoder used in the Neural Rewriting System. Specifically, we use Label-based Positional Encodings to enable the Selector to identify leaf formulas within very long sequences, and we constrain the receptive field of the self-attention layer to obtain localized attention on the close neighbors of each token. In the FastNRS, the Selector is designed to solve a text-segmentation task. The use of an encoder and the formalization of the selection problem as a text-segmentation one allows the parallelization of a core step of the algorithm to solve formula simplification problems. Formally, the Selector implements the function $multisel : F \rightarrow F^{L^n}$ where F^{L^n} represents the n-ary cartesian product of the set of leaf formulas F^L. The function maps a formula to one or more leaf formulas within it, corresponding to the left-hand sides of applicable rewrite rules.

In this implementation, given a sequence of tokens, the transformer performs a binary classification task on each token independently. A positive label indicates that a token is part of a leaf formula and will be selected for rewriting, while a negative label marks tokens that will not be selected. Given an input formula f, the Selector produces a mask over the input in which all parts of the formula that cannot be rewritten are masked. Using these masks, leaf formulas are extracted from the input, and a sequence of strings is obtained. Each string should correspond to the left-hand sides of some rewriting rule, and thus, it is given as input to the Solver, which computes the substitution according to the corresponding rule.

The Solver module

The Solver module in FastNRS shares the same architecture as the Solver module in the NRS and is designed to solve the same problem: applying the appropriate rewriting rule to compute the required substitutions. Also in this case, if the Solver outputs the ω symbol signaling the end of computation, the algorithm stops. Additionally, in FastNRS, we measure the confidence of the Solver's outputs, which plays a critical role in guiding the execution of the FastNRS Combiner module. This confidence measure helps ensure that only high-confidence outputs are used in the subsequent steps, enhancing the reliability of the overall system. We define the confidence of the Solver on any output e as the joint log-probability of sampling the output tokens from the distribution obtained by applying the softmax function to the logits produced by the final fully-connected layer of the decoder. Formally, $c_e = \sum_{i=1}^{N} \log(p_i)$, where N is the number of tokens in e, and p_i is the probability to sample token e_i in e.

The Combiner module

In contrast to the Neural Rewriting System (NRS), the Combiner module in FastNRS is not implemented using a neural architecture. Specifically, we do not use a convolutional neural network (CNN) to extract the position signal of the leaf formula identified by the Selector. Thanks to the text-segmentation implementation of the Selector, we can directly trace back the position of the identified leaf formula(s) within the input.

As a result, the Combiner module is implemented as a deterministic function, `cond_repl`, which takes the original formula f, the identified leaf formula f^L, its replacement $sol(f^L)$ computed by the Solver, and the measure of the Solver's confidence c_{e_k} as inputs. This confidence measure handles cases where the Solver output may contain errors. Indeed, despite the strong length generalization properties guaranteed by the modifications made to the Selector, minor defects in the segmentation of out-of-distribution sequences can still occur. Such defects could cause the extraction of corrupt parts of the input formula that do not correspond to the left-hand side of any rewriting rule, thus leading to meaningless substitutions computed by the Solver. Any f^L will thus be replaced with $sol(f^L)$ in f only if the corresponding measure of confidence of the Solver output c_{e_k} is sufficiently high. Intuitively, the measure c_{e_k} reflects the distance of the input from the training distribution of true left-hand sides of rewriting rules, and thus, it allows the identification of parts of the input that are not valid left-hand sides with sufficient degree of accuracy. In our experiments, we set the value of the confidence score threshold depending on the distribution of this quantity on the training samples, as detailed in Section 4.5.

If the input mask is drastically corrupted, and no e_k has a sufficiently high confidence score c_{e_k}, the computation is interrupted and this is considered an error of the model.

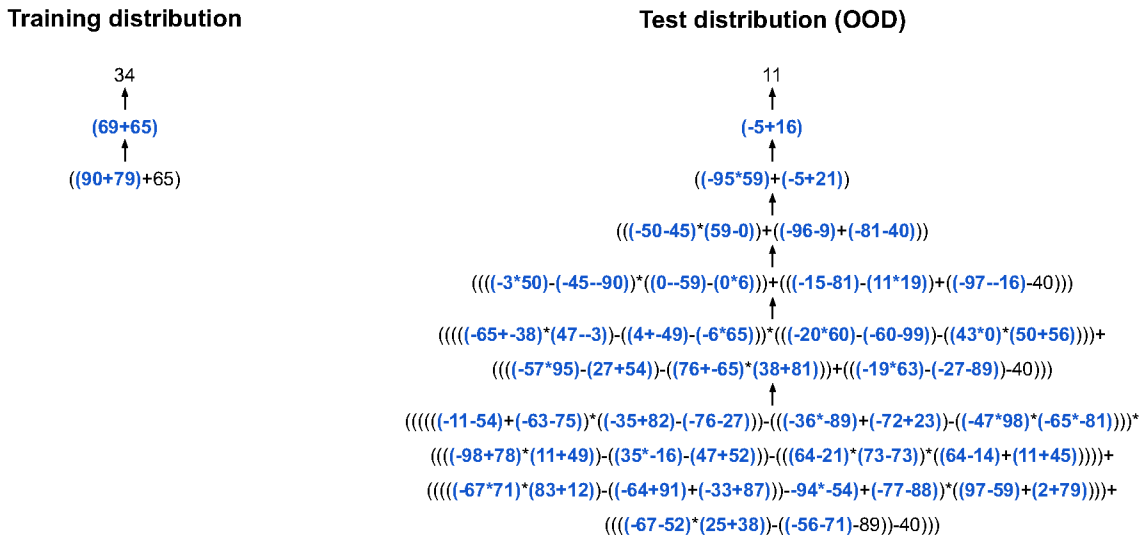


Figure 4.2: Visual representation of the simplification process of samples from the training set and the out-of-distribution (OOD) test set. The input parts that are simplified at each step are highlighted in blue.

4.4 Experimental Setup

4.4.1 Datasets

We benchmark the proposed architecture on four formula simplification problems from different domains: logic, operations on lists, arithmetic and algebra. For all problems, formulas are generated automatically, and their difficulty is determined by specifying the desired nesting level of the formula. Any formula is nested at each level in two points: exactly two arguments in the formulas on that level will be other formulas. We now describe the formulas in each domain in greater detail.

Logic

We build a dataset of nested logical formulas where the logical operators `AND`, `OR` and `NOT`, are applied on non-grounded literal variables, represented by lowercase letters in $\{\texttt{a}, ..., \texttt{z}\}$ or grounded logical variables `True` and `False`. Logical formulas are generated automatically specifying the desired number of nesting levels of a formula. Differently from the other three datasets, logical formulas are nested up to 12 times, thus requiring more steps to be solved. Each logical formula can be reduced either to a non-grounded literal variable or to a logical value between `True` and `False`. For example, the logical formula `(((z OR (z OR (b AND False))) OR z) AND ((((j OR False) AND True) AND False) OR True))` is nested 5 times, contains the literal variables `b`, `j` and `z`, the logical values `True` and `False`, and evaluates to `z`.

Listops

The ListOps dataset [71] was designed to assess neural networks' ability to construct parse trees for nested formulas. Initially, the dataset featured formulas with operations on integer lists, such as minimum, maximum, median, and sum modulo 10. We adapted the ListOps dataset to ensure that each nesting level had exactly two nesting points and allowed for specifying the number of arguments at any level. Focusing on the system's ability to generalize on deeply nested formulas rather than mastering specific operations, we limited the operations to minimum, maximum, and sum modulo 10.

Arithmetic

We created formulas using sum, subtraction, and multiplication operations between two integers sampled from the interval $[-99, 99]$. Since this study does not explore the ability to generalize to numbers with more digits than those encountered during training, we applied modulo 100 to the intermediate results obtained throughout the solution process.

Algebra

We focused on a subset of algebraic formulas that can always be deterministically simplified to a minimal form. These formulas consist of sums and subtractions between two monomials, with the final value always being a monomial. The numerical coefficients of the monomials were sampled from the interval $[-99, 99]$, and each monomial could include up to four literal variables chosen from $\{a, b, x, y\}$. All monomials in a given formula shared the same literal variables. Similar to the Arithmetic problem, all intermediate numerical values were computed modulo 100 when determining the formula's final value.

4.4.2 Models

Neural Rewriting Systems

We describe here how we built the training and validation sets for the Selector and Solver modules for both the NRS and the FastNRS. A visual representation of the solution process of samples from the training and test distributions is shown in Figure 4.2. Further methodological details can be found in Appendix B.1 and B.2.

In the training set of the Selector module, we included formulas with nesting levels of 1, 2, and 3 for all problems, along with atomic elements representing the final value of the initial formula. Simplifying formulas iteratively by reducing leaf formulas generates several intermediate simplifications of the original formula. To demonstrate the full solution process to the Selector, we also included these intermediate formulas as steps in the training set.

We created a separate in-distribution validation set with samples mirroring the structural characteristics of those in the training set. Unlike typical machine learning tasks, where models are tested on the same data distribution they were trained on, we aim for the Selector to demonstrate out-of-distribution (OOD) generalization abilities, identifying leaf formulas even in longer inputs than those encountered during training. Therefore, we also developed a distinct OOD validation set featuring formulas with higher structural complexity, using this

set for model selection. For all problems, we included in this set samples with nesting levels of 4, 5, and 6. To choose the most capable model throughout the iterative resolution process, we also added formulas representing examples of intermediate resolution steps. To manage the structural complexity of the formulas, we balanced the OOD validation sets across the nesting levels of the leaf expressions.

The training and validation sets for the Solver contained two types of samples: leaf formulas, which were mapped to their equivalent atomic elements, and atomic elements, which were mapped to the end-of-computation special symbol ω. To prevent bias toward solving leaf formulas, we generated training batches that included both types of samples with equal probability.

For the Selector, we adopted a problem-dependent tokenizer whose vocabulary contains atomic values, operators and parentheses for any problem. For example, the vocabulary for the arithmetic problem contains one token for each single- or double-digit integer, tokens for the sum, subtraction and multiplication operators and tokens for open and closed parentheses. In preliminary experiments, we also tried using a character-level tokenizer but observed worse out-of-distribution generalization capabilities of the Selector in some domains. For the Solver, we used a simple character-level tokenizer for all problems.

Both the NRS and the FastNRS behavior can be modulated by choosing the value of some hyperparameters at inference time. As described in Section 4.3.1, the Dynamic Windowing mechanism in the NRS Selector is regulated by a threshold T, which is used to determine on which formulas the mechanism should be applied. For both the NRS and the FastNRS, we define these thresholds as 150 for ListOps and algebraic formulas, and 125 for arithmetic formulas, while we do not employ the mechanism on formulas in the Logic domain. We select these thresholds by examining the average Selector confidence score for inputs of the same length, and choose the value corresponding to a decrease in average Selector confidence. We notice that this is always considerably higher than the maximum length of formulas observed during training. We measure the average Selector confidence on several formulas of different lengths and nesting levels drawn from both the in-distribution and out-of-distribution validation sets. We report a representation of the average confidence score values in Appendix C.2.

The conditional replacement of leaf formulas in the FastNRS operated by the `cond_repl` function is regulated by a threshold on Solver confidence. For both the single- and multi-domain versions of the FastNRS, we determined these thresholds based on the distribution of these scores on training samples, as mentioned in Section 4.3.2. Specifically, the thresholds used were -6 for ListOps, -2 for Arithmetic, -3 for Algebra, and -0.005 for Logic. The distributions of Solver confidence scores on training samples, which informed these thresholds, are provided in Appendix C.1.

Neural Data Router

The Neural Data Router (NDR) [70] is a modified transformer encoder designed to tackle algorithmic problems with robust out-of-distribution generalization capabilities. We refer the reader to Section 2.2.3 for a detailed technical description of the architecture.

This model has been previously tested on relatively simple algorithmic benchmarks, such as solving formulas in the original ListOps dataset and handling basic arithmetic formulas

with single-digit integers, which closely resemble the problems we address. However, the key difference in our Arithmetic and Algebra problems is the increased complexity of the operands. Additionally, we employ significantly fewer and less complex samples during training, as the NDR was initially trained on arithmetic and ListOps formulas containing up to 5 nested operations.

We made a minor modification to the architecture to adapt the model to our specific problems. In the original work, the problems always resulted in a single-digit integer, which the model was trained to output as the first token in the sequence generated by the encoder. Since this is not generally applicable to our problems, we read the final answer from the first k positions of the sequence produced by the encoder, where k is the maximum length of a problem's targets. As done with the Selector, we employ a problem-dependent tokenizer at the atomic value level when training the Neural Data Router. Therefore, k equals 3 in the case of Algebra, 2 for Arithmetic and 1 for ListOps and Logic problems.

We constructed all development sets for the NDR using the same top-level formulas included in the analogous sets for the Selector. Following the original experimental protocol, we ensured that the training set was balanced across nesting levels. Similar to the Selector module training, we created both in-distribution and out-of-distribution validation sets, using the latter to optimize hyperparameters through a Bayesian search using the Weights & Biases platform in the same hyperparameters intervals described in the original work. The final hyperparameter values for each task are detailed in Appendix D.

OpenAI GPT-4

The most effective method found by researchers and practitioners to improve the reasoning capabilities in large language models is to use specialized prompts. Specifically, Chain-of-Thought (CoT) prompting has been found to enhance the performance of large language models on reasoning tasks by facilitating step-by-step solution procedures. We opted to prompt GPT-4 using a combination of self-consistency prompting [109] and zero-shot Chain-of-Thought (CoT) [107]. Zero-shot CoT is a simpler alternative to traditional CoT prompting, achieving comparable performance on reasoning benchmarks without the need to engineer exemplars for few-shot reasoning. This is done by simply initiating the model's response with the sentence: "Let's think step-by-step." Once the model generates a response, it is prompted again to produce a well-formatted output. Self-consistency prompting leverages the idea that reasoning problems can have multiple valid paths leading to the same conclusion. Thus, we generate 10 outputs for each input and select the most consistently produced one. This approach enhances confidence in the model's output and significantly improves accuracy, leading to a marked improvement in performance.

The zero-shot CoT prompt was designed by providing the model with a brief description of the problem and then asking it to solve it. For instance, the zero-shot CoT prompt for the ListOps input sample `[MIN[SM54][MIN39]]` is: "*`MIN`, `MAX`, and `SM` are operators on lists of single-digit integers, representing minimum, maximum, and sum modulo 10, respectively. Solve the following expression using these operators: `[MIN [SM 5 4] [MIN 3 9]]`.*" Following this initial prompt, the model was subsequently prompted a second time to provide a well-formatted final answer.

OpenAI o1-preview

The OpenAI o1 model is a recently developed model designed to significantly enhance complex reasoning tasks. It builds on previous models like GPT-4 but focuses on spending more time "thinking" before responding. This makes it particularly suitable for domains such as mathematics, coding, and reasoning problems. The o1 series introduces several new features, including "reasoning tokens," which are assumed to represent the model's internal thought process. These proved to be especially useful for tasks where critical thinking and reasoning are required. There are currently two versions of the model available: the o1-preview and a smaller, faster, and more cost-effective version called o1-mini. While o1-mini is designed for efficiency in coding and math tasks, the o1-preview model excels in broader knowledge areas and complex reasoning challenges.

4.5 Results

In this section, we present the evaluation of the Neural Rewriting System and the Fast Neural Rewriting System, focusing on performance and efficiency, both in single-domain and multi-domain scenarios. Since the final target of any formula corresponds to an atomic value, we measure the performance of the models on all tasks using Sequence Accuracy, i.e. the exact match between model output and target sequence. The test sets on which all models are evaluated are composed of 100 formulas per nesting level. We observed non-significative variance across runs, which we therefore do not report.

4.5.1 Learning domain-specific convergent term rewriting systems

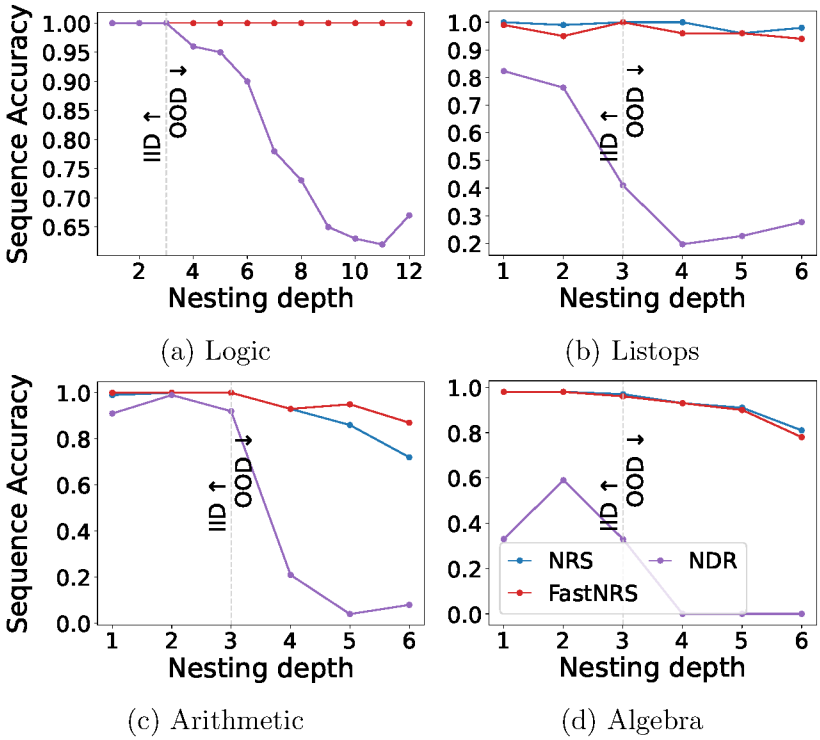
In this section, we evaluate the performance of both models across all four datasets — Logic, ListOps, Arithmetic, and Algebra — in a single-domain scenario. We compare the NRS and the FastNRS to another neural baseline the Neural Data Router (NDR), which has also been trained on individual tasks separately.

The performance of the models on all domains is represented in Figure 4.3. Across all datasets, the models show similar performance on in-distribution samples, with NDR generally performing the worst. However, on out-of-distribution samples, the baseline models exhibit a much sharper decline in performance compared to both the NRS and the FastNRS.

Interestingly, the FastNRS is more accurate than the NRS on deeply nested arithmetic formulas. Therefore, in this case, the design choices in the FastNRS yield a significant improvement in terms of performance other than efficiency. By examining the type of errors committed by both systems on arithmetic formulas in Section 4.5.4, we will clearly see how the superior performance of the FastNRS depends on the greater robustness in the identification of leaf formulas, guaranteed by the text segmentation-based implementation of the Selector module.

4.5.2 Learning multi-domain convergent term rewriting systems

As detailed in Section 4, the architectures and execution dynamics of both the NRS and the FastNRS are specifically designed to support learning algorithms within the class of conver-



(a) Logic (b) Listops

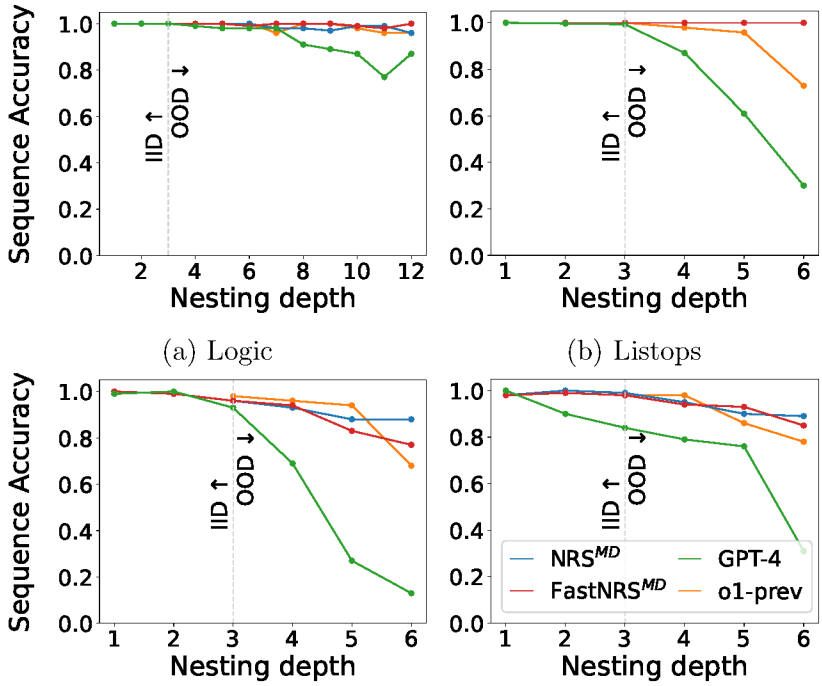
(c) Arithmetic (d) Algebra

Figure 4.3: Performance of FastNRS, NRS, and NDR on each domain. Sequence accuracy is measured on data splits of 100 samples.

gent term rewriting systems. This capability is primarily attributed to two key aspects: the algorithmic-inspired modular design of the architectures and the strong out-of-distribution generalization capability of the Selector module (see Section 4.5.5). As we mentioned, this generalization capability is particularly useful in a multi-domain scenario.

In this case, we choose to benchmark the models against OpenAI's GPT-4 and o1-preview, whose training regimen is multi-domain by definition. We notice that o1-preview was benchmarked only on out-of-distribution data splits, as simpler formulas can be considered trivial for this type of model. Performance metrics for the four models are illustrated in Figure 4.4. GPT-4 is the weakest performer across the tasks, particularly struggling with the ListOps, Arithmetic, and Algebra benchmarks. While it still surpasses the previously proposed Neural Deductive Reasoner (NDR), it shows significant limitations when faced with deeply nested formulas with complex operands. On the other hand, o1-preview demonstrates a substantial leap in performance over GPT-4. This improvement can be attributed to the more advanced CoT reasoning process implemented in the model, which enables it to process and simplify formulas more effectively, even within the complex tasks under consideration. The model's results could provide evidence of the importance of producing explicit reasoning steps when addressing tasks that demand compositional reasoning.

In the multi-domain training scenario, both the Neural Rewriting Systems demonstrate their capacity to generalize to out-of-distribution samples. Specifically, the NRS has a slightly better performance than the FastNRS, especially on out-of-distribution samples, demonstrating the effectiveness of the specialized architectural elements introduced for this purpose. In



(a) Logic (b) Listops

(c) Arithmetic (d) Algebra

Figure 4.4: Performance of multi-domain models: GPT-4, o1-preview, FastNRS and NRS on each domain. Sequence accuracy is measured on data splits of 100 samples.

the Logic and ListOps domains, the FastNRS maintains nearly the same accuracy as the NRS, with only a slight accuracy decrease of few percentage points in some cases. Notice that in the Logic domain, we evaluated the models on out-of-distribution test formulas with up to 12 nesting levels, where both models achieve consistently high accuracy. In these domains, the models show superior or similar performance to o1-preview on both in-distribution and out-of-distribution data splits. In the more complex Arithmetic and Algebra domains, there is a slightly larger drop in accuracy on certain out-of-distribution formulas. On these two tasks, the o1-preview model shows superior performance on some of the out-of-distribution splits. However, the NRS still outperforms o1-preview on the most complex formulas, demonstrating its effectiveness in learning convergent term rewriting systems with significant generalization capabilities.

Problem	**# Param.**	**Inf. time**
Multi-domain	18,651,574	51h 48m 7s
Logic	3,047,365	16m 57s
ListOps	3,842,209	8h 24m 30s
Arithmetic	10,890,364	7h 59m 30s
Algebra	9,904,856	14h 16m 48s

Table 4.1: Space and time efficiency statistics for the NRS.

Problem	**# Param.**	**Inf. time**
Multi-domain	15,061,616	3m 42s
Logic	2,501,795	38s
ListOps	4,095,633	32s
Arithmetic	8,728,338	50s
Algebra	8,752,920	3m 02s

Table 4.2: Space and time efficiency statistics for the FastNRS.

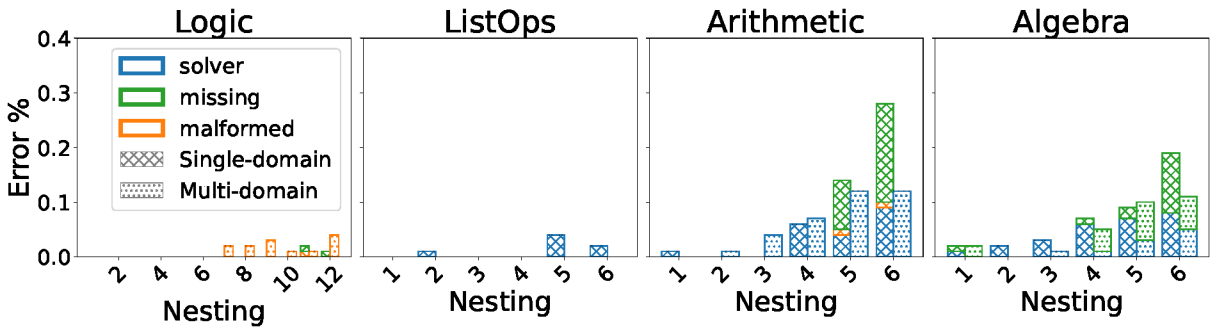


Figure 4.5: Breakdown of NRS errors by type in single- and multi-domain settings.

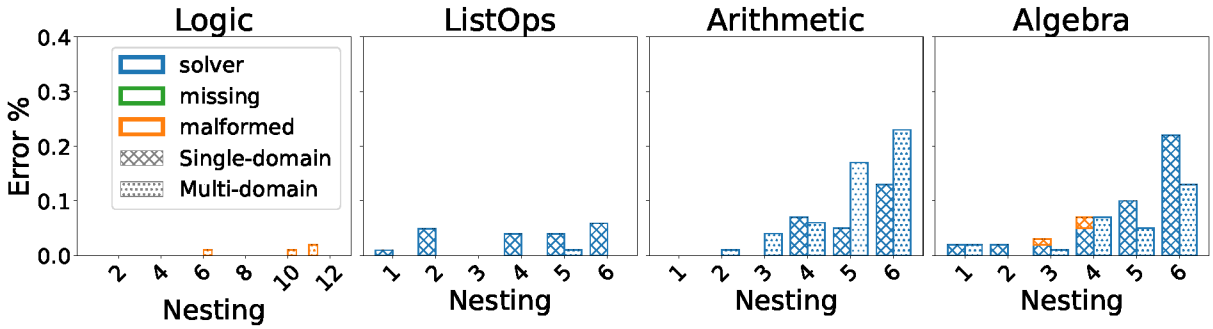


Figure 4.6: Breakdown of FastNRS errors by type in single- and multi-domain settings.

4.5.3 Analysis of efficiency

The design of the FastNRS mainly leads to performance improvements with respect to the twin implementation of the framework we propose. The number of parameters and inference time statistics for the NRS and the FastNRS in both training scenarios are reported in Table 4.1 and Table 4.2.[2] Importantly, the FastNRS has, in most cases, fewer parameters than the NRS. Moreover, it achieves several order of magnitude reduction in inference time, being hours faster at test time, both in the case of single- and multi-domain models. In the case of the multi-domain setting, cumulative inference time on all test samples in the four tasks is reported for both models. This efficiency gain demonstrates the possibility of implementing an efficient learning mechanism for the solution process of symbolic formulas across different domains within a single, unified neural circuitry. The modifications introduced in the FastNRS thus not only improve computational efficiency but also preserve the model's generalization capabilities, and, as we have seen, can occasionally improve performance.

4.5.4 Analysis of errors

We analyze the errors committed by the Neural Rewriting Systems when simplifying mathematical formulas in the four domains. We consider both the single-domain and multi-domain training settings. We breakdown errors for each domain by error type, and visualize the

[2]All runs were executed on a single NVIDIA A100 GPU. Statistics reported for the NRS refer to the simplest hyperparameters configuration that achieves the best performance in each test scenario.

analysis in stacked barplots. We consider three error cases: the one in which leaf formulas identified by the Selector are not present in the input formula (`missing`), the case in which they are present but they are not valid formulas (`malformed`), and the case in which they are solved incorrectly (`solver`).

We start by observing that there are no errors in the `missing` class in the case of FastNRS, which is expected given that we use a segmentation-based approach to the Selector task. On the other hand, this type of error represents the majority of those committed by the Selector in the NRS, while errors in the `malformed` class are quite rare for both the NRS and the FastNRS. Therefore, we can conclude that when Selector modules in both architectures identify leaf formulas that are present in the input, these tend to be well-formed.

The multi-domain setting reveals interesting positive effects, but also introduces challenges for both the NRS and FastNRS. In the case of NRS, multi-domain training seems to mostly have a neutral or positive effect across all tasks, significantly improving performance on algebraic and arithmetic formulas and reducing the amount of `missing` errors on the latter (see Figure 4.5). It seems that by training the Selector on multiple tasks, the NRS becomes more adept at identifying valid leaf formulas, reducing the number of this type of error.

The effects of multi-domain training on FastNRS are more heterogeneous and depend on the specific task. In the ListOps and algebra domains, the multi-domain model shows consistent improvements, with a marked reduction in Solver errors. Interestingly, FastNRS performance worsens in the multi-domain setting on arithmetic formulas, where Solver errors increase. This could reflect the fact that arithmetic formulas, involving operations between double-digit integers, are the hardest type of operation for the Solver.

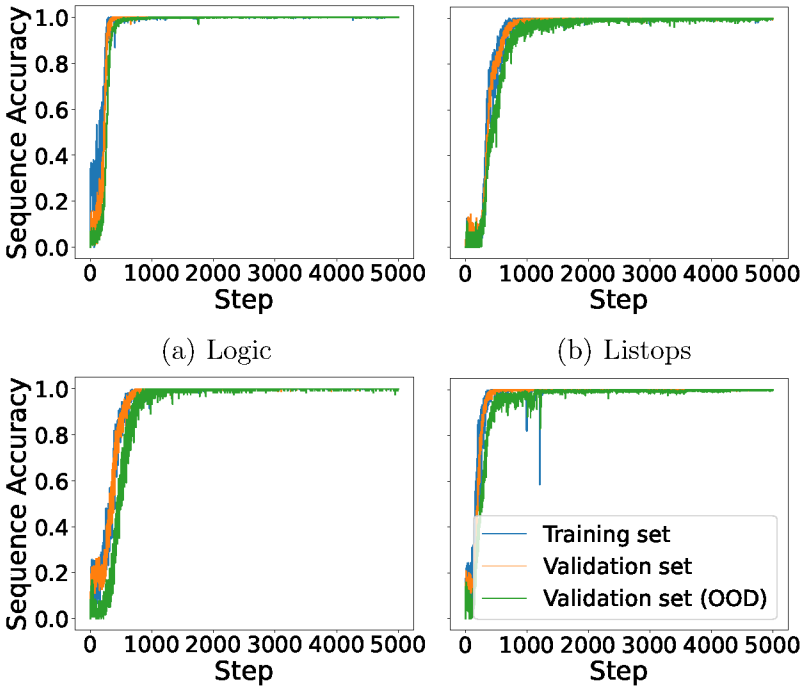
Notably, in both single- and multi-domain scenarios, FastNRS never fails to find at least one valid leaf in any iteration, indicating robustness in the Selector module.

4.5.5 Out-of-distribution generalization in the FastNRS Selector

As we described in Section 4.3.2, the Selector module is a transformer encoder with two main architectural modifications: Label-based Positional Encodings and a strong limitation of the self-attention's receptive field. Furthermore, we described how designing the Selector as a text segmentation module allowed us to simplify and improve the efficiency of the whole architecture.

The plots in Figure 4.7 show how a Selector with the abovementioned architectural modifications, trained to segment input formulas, exhibits almost indistinguishable convergence trends on in- and out-of-distribution instances in all problems. Carefully tuned Selector modules can thus segment an input formula several tens of tokens longer than formulas observed during training, with very limited or zero error rate. Additional results about the impact of the width of the Selector's self-attention window and the number of layers in the model are reported in Appendix B.2.1.

We also notice that the Selector is particularly sample efficient, as it requires only $5,000$ iterations to converge to almost perfect accuracy, while the best NRS Selector modules, chosen after hyperparameters tuning, could be trained for up to $30,000$ iterations, depending on the task (see Appendix B.3). We also noticed that in all domains except ListOps, FastNRS Selector modules are several thousands of parameters smaller than their NRS counterparts



(a) Logic (b) Listops

(c) Arithmetic (d) Algebra

Figure 4.7: FastNRS Selector accuracy during training on the text segmentation task.

in the same domain (not shown here), and we observed that further increasing the number of parameters did not affect performance.

4.6 Limitations

Despite the significative generalization capabilities demonstrated by the framework we propose, its scope and applicability is constrained by several limitations. First, the current implementation of NRS is restricted to tasks that can be framed as sequence-based rewriting problems. This assumption limits the range of tasks it can handle: for example, many real-world tasks involve hierarchical structures or visual reasoning, which cannot be addressed within the current sequence-only framework. Second, rewriting rules must currently operate on local substrings of the input sequence, even in the FastNRS which identifies multiple substrings in parallel. Finally, the NRS is built on an algorithmic structure where the steps of the rewriting process are predefined by the human designer. While the system efficiently applies these predefined rules generalizing to more complex, unseen cases, it does not possess the capacity to learn or infer the rewriting algorithm from data. Therefore, this design choice limits the possibility of applying the system to new problems beyond the algorithmic template initially provided.

4.7 Conclusions

In this chapter, we addressed the three primary research questions guiding our investigation into neuro-symbolic architectures for convergent term rewriting systems. First, we proposed a hybrid neuro-symbolic framework specifically designed for this class of problems. We identify two key elements in the design of the framework: first, an algorithmic blueprint for a modular architecture that mirrors the structure of the rewriting algorithm itself; second, employing neural architectures that achieve strong out-of-distribution generalization capabilities to enable algorithmic-like behavior in modules that should process arbitrarily long inputs, such as the Selector in the Neural Rewriting Systems.

We introduced two implementations of the framework we propose: the Neural Rewriting System (NRS) and the Fast Neural Rewriting System (FastNRS). Both models were built to address the core question of how to design a system that can learn and generalize across the class of problems solvable by convergent term rewriting systems.

We have compared the Neural Rewriting Systems trained in a single-domain scenario with the Neural Data Router as a representative of small-scale neural architectures specialized to learn single reasoning tasks. Our systems clearly outperformed the baseline, especially on out-of-distribution samples. We further compared both systems trained in a multi-domain scenario against two general-purpose Large Language Models: OpenAI's GPT-4 and the recently presented o1-preview model designed to solve complex reasoning tasks. Although the performance of our models on the most complex formulas consistently surpassed that of GPT-4 on the same problems, o1-preview showed surprising capabilities of solving even very complex formulas with a relatively high degree of accuracy. While o1-preview outperformed the Neural Rewriting Systems on groups out-of-distribution of formulas of intermediate complexity in some tasks, the models we propose consistently achieved an equal or higher accuracy on the hardest formulas in all tasks, demonstrating a significant generalization capability. The drop in performance of o1-preview on complex problems might suggest a fundamental lack of systematic reasoning capabilities and understanding of mathematical concepts still persistent in this new class of models, as also noted in recent work by Mirzadeh et al. [142].

These results show that both systems are capable to solve multiple types of problems within a unified framework, particularly when trained in multi-domain settings. Considering these results, we can thus address the second research question guiding our investigation regarding the impact of design choices on performance. The superior performance of the Neural Rewriting System on the most complex formulas and tasks demonstrated the effectiveness of the specialized error-correction mechanisms employed in the architecture. At the same time, the design choices introduced in the FastNRS to improve the efficiency of the system proved not to hinder performance significantly.

We presented an empirical analysis of the memory and time efficiency of both systems, providing evidence of the impact of the design choices on these aspects, thus addressing our third research question. The analysis showed that the specialized mechanisms in the Neural Rewriting System significantly impact time efficiency. At the same time, the key modifications introduced in the FastNRS—specifically the text-segmentation-based implementation of the Selector module—proved critical in improving memory efficiency, training time, and

inference speed, without sacrificing performance.

The strengths of our architecture can be traced back to its modular design, which is informed by the rewrite algorithm, and to the architectural modifications to the transformer, which have proven effective in enabling strong out-of-distribution generalization. However, these design choices also limit the scope of applicability of our system to sequence-based problems solvable by convergent term rewriting systems with local substitution rules. Future research could be dedicated to expanding the system to handle rules that could act on patterns across different parts of the sequence. This would involve rethinking the Selector, where it would be necessary to design a neural circuit that is capable of generalizing on the selection of non-local patterns as consistently as the current circuit does with local ones. Furthermore, the substitution mechanism should be designed to be capable of reliably replacing the global patterns, potentially maintaining a level of flexibility to noise and errors in the Selector output.

As we previously noticed, the algorithmic-informed design of our systems, which guarantees robustness, is defined *a priori* rather than being learned from data, and thus limits their scope of applicability. Designing an end-to-end learned system where the algorithmic blueprint of the problem at hand is inferred directly from data could prove to be a challenging and interesting venue for future research. While defining the algorithmic blueprint for a specific class of problems in advance imposes on the system a strong inductive bias that is aligned to the class itself, a general-purpose framework for algorithmic learning would involve designing a learning bias that allows the system to *dynamically align* to the specific class of problems under consideration.

Chapter 5

Conclusions

In this thesis, we offered several contributions to the research areas concerned with learning systematic reasoning procedures from data. We now shortly reconsider the motivation for this work. Then, we return to the research questions that guided our investigation and provide answers based on the evidence presented in the previous chapters. Finally, we acknowledge some existing limitations of our findings and remaining open challenges.

5.1 Scope and motivation

The field of AI has historically been divided into two dominant paradigms: learning-based AI, represented by machine learning and deep learning systems, and reasoning-based AI, including rule-based and formal reasoning systems. The former has excelled at tasks like image recognition and language modeling, where vast amounts of data allow systems to learn patterns and generalize to new inputs. However, despite its successes, learning-based AI struggles with tasks that require structured, logical reasoning—especially in situations where training data is limited or where generalization beyond surface-level patterns is required. In contrast, classical reasoning-based AI, which emerged from the symbolic AI tradition, focused on explicit rule-based systems capable of reasoning abstractly. These systems were designed to manipulate symbols and rules in well-defined domains, excelling in tasks that required logical inference, such as mathematical theorem proving or expert systems. However, they lacked the ability to scale to more complex, data-rich environments or adapt to new tasks without significant manual intervention.

A central challenge in modern AI is to combine the strengths of both approaches: creating systems that can learn reasoning patterns from data, thus ideally keeping both the flexibility and effectiveness of learning-based systems, and the reliability of reasoning-based systems. This would allow for AI systems that can reason abstractly in unfamiliar situations, solve problems in unseen domains, and adapt to new tasks.

5.2 Addressed research questions and contributions

In this thesis, building on the tradition of cognitive science and artificial intelligence and aligning with a recent body of research presented in Chapter 2, we considered the concept

of compositionality as a foundational concept that can be used to assess the systematic reasoning capabilities of learning-based neural AI systems.

Specifically, we focused on the problem of simplifying nested mathematical formulas, which possess clear hierarchical features, a key property of compositionality, identified by scholars in both reasoning processes and languages, both formal and natural. Such a problem was a common thread in our investigations, serving both as a testbed to assess the reasoning and generalization capabilities of existing architectures and as a class of problems on which we focused our efforts in designing learning-based neuro-symbolic systems with strong generalization capabilities.

We critically considered one of the prevailing perspectives in current machine learning and artificial intelligence research, that looks at auto-regressive large-language models as a promising paradigm to achieve reliable systematic reasoning. Specifically, we took into account claims according to which reasoning capabilities in large-language models emerge as a result of model scale and specialized prompting methods. Based on the experimental evidence and analyses presented in Chapter 3, we could thus answer the three research questions posed at the beginning of this thesis:

1. Does the capability to reason systematically emerge as the model scale increases?

We find that both scaling and fine-tuning models in the Llama 2 family do improve their average performance on ListOps and arithmetic formulas. However, breaking down the analysis on groups of formulas of equal complexity, scaling the model only improved performance on relatively simple problems for both vanilla and fine-tuned models. Therefore, we cannot consider either scaling or fine-tuning reliable ways to make systematic reasoning capabilities emerge in Llama 2 models.

2. Can this capability be reliably elicited through careful prompting?

We find that specialized prompting methods designed to improve reasoning capabilities, and, specifically, Chain-of-Thought-like prompting, improve the overall performance of GPT models on ListOps, arithmetic and algebraic formulas. However, by analyzing the improvement brought by the methods on formulas of the same complexity, we observed that even the most effective methods mostly improved performance on simple data formulas, and could therefore not elicit in these models reasoning procedures with algorithmic-like generalization properties.

3. Overall, can we say that LLMs genuinely learn to reason systematically?

Scaling, fine-tuning and in-context learning through prompting are considered the most effective ways to improve performance of LLMs on some task. Given the evidence considered above, and the conclusions drawn from it, we can say there is currently no evidence that LLMs learn to reason systematically in a consistent and reliable way.

The second part of this thesis contributes to the research area involved with the design and implementation of artificial intelligence systems that can learn to reason. Our proposal is made from a perspective that is closely aligned both with research on neuro-symbolic artificial intelligence systems, described in Section 1.1, and with research on neural networks that can learn algorithms from data, overviewed in Section 2.3. In Chapter 4, we formalize the problem of solving nested mathematical formulas, showing it can be solved by class of algorithms implemented by convergent term rewriting systems. Starting from this formalization, we address the first research question posed in Chapter 1:

1. How can a hybrid neuro-symbolic architecture be designed specifically for the class of problems we consider?

We propose a neuro-symbolic framework with two key design elements: first, its modular design is aligned with the steps of the rewriting algorithms that should be learned by the system; second, the Selector, a neural module that should process arbitrarily long inputs, should possess strong out-of-distribution capabilities. These elements allow the framework to be implemented in actual systems that possess strong generalization properties on the class of problems solved by convergent term rewriting systems.

2. What is the impact of different design elements on the system's performance?

We implement the proposed neuro-symbolic framework in two different systems: the Neural Rewriting System (NRS) and the Fast Neural Rewriting System (FastNRS). The two mainly differ in the implementation of the Selector module, which contains specialized mechanisms to improve accuracy in the case of NRS, and has a text segmentation-based implementation in the case of FastNRS. Both systems could be trained to solve formulas in a single domain, or in multiple domains (four, in our experiments) at the same time. We present empirical evidence on the performance of two systems, benchmarking them against the Neural Data Router, OpenAI's GPT-4 and o1-preview models.

Based on this evidence, we conclude that the design choices introducing specialized error-correction mechanisms in the Neural Rewriting System were effective in achieving high accuracy on the solution of the most complex formulas and tasks, on which our model consistently outperformed the baselines. At the same time, we note that the text segmentation-based implementation of the FastNRS Selector did not affect significantly its performance on complex formulas.

3. What is the impact of these design choices on efficiency, measured in terms of training time, inference time, and memory consumption?

An analysis of the time and space efficiency of both systems showed that the specialized error correction mechanisms in the Neural Rewriting System significantly increase inference time. At the same time, the text segmentation-based implementation of the FastNRS Selector proved particularly effective in improving memory efficiency, training time, and inference speed.

5.3 Limitations and open challenges

The experiments, analyses, and arguments presented in this thesis are not without limitations. Our findings suggest that large language models lack consistent systematic reasoning capabilities. However, these results are constrained to the specific models and prompting techniques we evaluated, in a rapidly evolving field where new models and methods are introduced almost daily. In particular, while we cannot claim that OpenAI's o1-preview demonstrates reasoning abilities on par with classical algorithms in tasks like mathematical formula simplification, it remains difficult to draw definitive conclusions about the potential of the new training paradigm introduced by o1. The continuous development in this area will surely leave the space of investigation on the reasoning capabilities of these models open for future research.

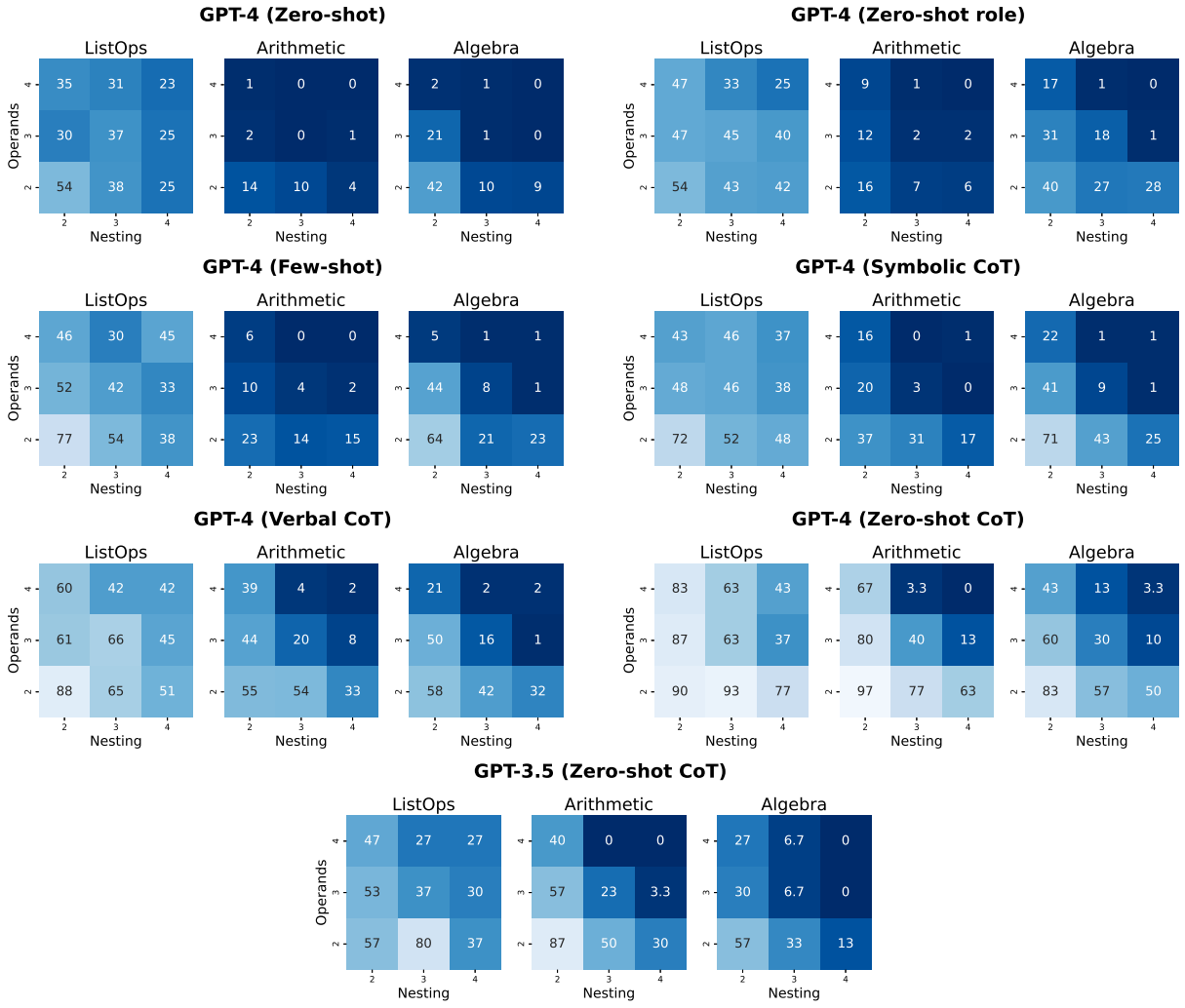
As discussed in Chapter 4, the neuro-symbolic framework we propose also has limitations. First, its application is restricted to sequence-based problems that can be solved using convergent term rewriting systems with local substitution rules. Second, the algorithmic design of our system is predetermined rather than learned from data, which narrows its applicability to problems that fit this predefined structure. More generally, our results indicate that achieving generalization in learning-based systems requires precise problem characterization—meaning the problem must be formalizable and reflected in the system's architecture.

Despite these constraints, our findings are promising. They suggest that designing specialized systems capable of learning systematic reasoning from data is a feasible approach for a potentially large class of problems. Developing a neural or neuro-symbolic framework where the algorithm itself is learned from data remains an intriguing direction for future research, towards the greater goal of artificial intelligent systems that can learn reliable systematic reasoning from experience.

Appendix A

Additional GPT benchmarking results

A.1 Other models and prompting methods



A.2 Examples of prompts

A.2.1 ListOps

Prompt type	Example
Zero-shot	Q: MIN, MAX and SM are operators on lists of single-digit integers which have the semantics of minimum, maximum and sum modulo 10, respectively. Solve the following expression involving these operators: [MIN[MAX[MIN68]8][MAX[SM23]6]]. A: The final result is (arabic numeral):
Zero-shot role	Q: MIN, MAX and SM are operators on lists of single-digit integers which have the semantics of minimum, maximum and sum modulo 10, respectively. Solve the following expression involving these operators: [MIN[MAX[MIN68]8][MAX[SM23]6]]. A: The final result is (arabic numeral):
Few-shot	Q: MIN, MAX and SM are operators on lists of single-digit integers which have the semantics of minimum, maximum and sum modulo 10, respectively. Solve the following expression involving these operators: [MIN37]. A: [MIN37]=3. Q: MIN, MAX and SM are operators on lists of single-digit integers which have the semantics of minimum, maximum and sum modulo 10, respectively. Solve the following expression involving these operators: [MAX[MIN41]2]. A: [MAX[MIN41]2]=2. Q: MIN, MAX and SM are operators on lists of single-digit integers which have the semantics of minimum, maximum and sum modulo 10, respectively. Solve the following expression involving these operators: [SM[SM794][SM498]7]. A: [SM[SM794][SM498]7]=8. Q: MIN, MAX and SM are operators on lists of single-digit integers which have the semantics of minimum, maximum and sum modulo 10, respectively. Solve the following expression involving these operators: [MIN[MAX[MIN68]8][MAX[SM23]6]].

Symbolic CoT	Q: MIN, MAX and SM are operators on lists of single-digit integers which have the semantics of minimum, maximum and sum modulo 10, respectively. Solve the following expression involving these operators: [SM73]. A: [SM73]= 0. Q: MIN, MAX and SM are operators on lists of single-digit integers which have the semantics of minimum, maximum and sum modulo 10, respectively. Solve the following expression involving these operators: [SM[SM86]1]. A: [SM[SM86]1]= [SM41]= 5. Q: MIN, MAX and SM are operators on lists of single-digit integers which have the semantics of minimum, maximum and sum modulo 10, respectively. Solve the following expression involving these operators: [MIN[MAX243]4[MAX937]]. A: [MIN[MAX243]4[MAX937]]= [MIN[MAX243]49]= [MIN449]= 4. Q: MIN, MAX and SM are operators on lists of single-digit integers which have the semantics of minimum, maximum and sum modulo 10, respectively. Solve the following expression involving these operators: [MAX[SM[SM85][MIN84]][SM3[MIN40]]].

Verbal CoT	Q: MIN, MAX and SM are operators on lists of single-digit integers which have the semantics of minimum, maximum and sum modulo 10, respectively. Solve the following expression involving these operators: [MIN82]. A: Let's solve the following expression: [MIN82]. Simplifying the expression, we get to the final result: 2 Q: MIN, MAX and SM are operators on lists of single-digit integers which have the semantics of minimum, maximum and sum modulo 10, respectively. Solve the following expression involving these operators: [MIN[SM56][MAX87]]. A: Let's solve the following expression: [MIN[SM56][MAX87]]. Simplifying an expression without nested parentheses, we get: [MIN[SM56]8]. Simplifying the expression, it becomes: [MIN18] Taking an immediate solution step, we get to the final result: 1. Q: MIN, MAX and SM are operators on lists of single-digit integers which have the semantics of minimum, maximum and sum modulo 10, respectively. Solve the following expression involving these operators: [MIN[MIN326]0[SM851]]. A: Let us recall the expression to be solved: [MIN[MIN326]0[SM851]]. By solving a simple expression, we obtain: [MIN[MIN326]04]. Solving a expression within a single pair of brackets, we get: [MIN204]. Taking an immediate solution step, we get to the final result: 0. Q: MIN, MAX and SM are operators on lists of single-digit integers which have the semantics of minimum, maximum and sum modulo 10, respectively. Solve the following expression involving these operators: [MIN[MAX[MIN68]8][MAX[SM23]6]].
Zero-shot CoT	Q: MIN, MAX and SM are operators on lists of single-digit integers which have the semantics of minimum, maximum and sum modulo 10, respectively. Solve the following expression involving these operators: [MIN[MAX[MIN68]8][MAX[SM23]6]]. A: Let's think step-by-step.

A.2.2 Arithmetic

Prompt type	Example

Zero-shot	Q: Solve the following arithmetic expression computing the modulo 100 of each intermediate value if it's positive, and the modulo -100 if it's negative: (-66-(-84*(-34+0))). A: The final result is (arabic numerals):
Zero-shot role	Q: Solve the following arithmetic expression computing the modulo 100 of each intermediate value if it's positive, and the modulo -100 if it's negative: (-66-(-84*(-34+0))). A: The final result is (arabic numerals):
Few-shot	Q: Solve the following arithmetic expression taking each intermediate value modulo 100 if it's positive, and modulo -100 if it's negative: (51*39). A: (51*39)=89. Q: Solve the following arithmetic expression taking each intermediate value modulo 100 if it's positive, and modulo -100 if it's negative: ((28*-53)*(-76*90)). A: ((28*-53)*(-76*90))=60. Q: Solve the following arithmetic expression taking each intermediate value modulo 100 if it's positive, and modulo -100 if it's negative: (40-54-(-33–97+-19)). A: (40-54-(-33–97+-19))=-59. Q: Solve the following arithmetic expression taking each intermediate value modulo 100 if it's positive, and modulo -100 if it's negative: (-66-(-84*(-34+0))).

Symbolic CoT	Q: Solve the following arithmetic expression taking each intermediate value modulo 100 if it's positive, and modulo -100 if it's negative: (-16*-37). A: (-16*-37)= 92. Q: Solve the following arithmetic expression taking each intermediate value modulo 100 if it's positive, and modulo -100 if it's negative: ((87*-51)-(47*-6)). A: ((87*-51)-(47*-6))= ((87*-51)–82)= (-37–82)= 45. Q: Solve the following arithmetic expression taking each intermediate value modulo 100 if it's positive, and modulo -100 if it's negative: ((-12–28-74)+-21+(76+-32+-87)). A: ((-12–28-74)+-21+(76+-32+-87))= ((-12–28-74)+-21+-43)= (-58+-21+-43)= -22. Q: Solve the following arithmetic expression taking each intermediate value modulo 100 if it's positive, and modulo -100 if it's negative: (((-33-39)*(67*65))-22).

Verbal CoT	Q: Solve the following arithmetic expression taking each intermediate value modulo 100 if it's positive, and modulo -100 if it's negative: (-14*88). A: Let us recall the expression to be solved: (-14*88). Simplifying the expression, we get to the final result: -32 Q: Solve the following arithmetic expression taking each intermediate value modulo 100 if it's positive, and modulo -100 if it's negative: ((92*26)*(-35*59)). A: Let us recall the expression to be solved: ((92*26)*(-35*59)). Simplifying the expression, it becomes: ((92*26)*-65) Solving a expression within a single pair of brackets, we get: (92*-65). Simplifying the expression, we get to the final result: -80 Q: Solve the following arithmetic expression taking each intermediate value modulo 100 if it's positive, and modulo -100 if it's negative: (83-(46+-5-54)-25). A: We need to solve the following expression: (83-(46+-5-54)-25). Taking an immediate solution step, we obtain: (83–13-25). As this expression is in a simple form, we can get to the final result: 71 Q: Solve the following arithmetic expression taking each intermediate value modulo 100 if it's positive, and modulo -100 if it's negative: (-66-(-84*(-34+0))).
Zero-shot CoT	Q: Solve the following arithmetic expression computing the modulo 100 of each intermediate value if it's positive, and the modulo -100 if it's negative: (((-33-39)*(67*65))-22). A: Let's think step-by-step.

A.2.3 Algebra

Prompt type	Example
Zero-shot	Q: Simplify the following algebraic expression, computing the modulo 100 of the numerical coefficient of each intermediate value if it's positive, and the modulo -100 if it's negative: ((-78*b*x*y+(+50*b*x*y+-22*b*x*y))+((-b*x+-57*b*x)+(-38*b*x+-99*b*x))). If possible, factor by grouping the final result. A: The final result is (algebraic expression):

Zero-shot role	Q: Simplify the following algebraic expression, computing the modulo 100 of the numerical coefficient of each intermediate value if it's positive, and the modulo -100 if it's negative: (-85*a*y+((-98*a*x*y+2*a*x*y)+0*x*y*a)). If possible, factor by grouping the final result. A: The final result is (algebraic expression):
Few-shot	Q: Solve the following algebraic expression taking the numerical coefficient of each intermediate value modulo 100 if it's positive, and modulo -100 if it's negative: (-55*b*x*y+-8*b*x). If possible, factor by grouping the final result. A: (-55*b*x*y+-8*b*x)=-b*x*(55*y+8). Q: Solve the following algebraic expression taking the numerical coefficient of each intermediate value modulo 100 if it's positive, and modulo -100 if it's negative: ((-54*x*y+-68*x*y)+(-99*x*y++62*x*y)). If possible, factor by grouping the final result. A: ((-54*x*y+-68*x*y)+(-99*x*y++62*x*y))=-59*x*y. Q: Solve the following algebraic expression taking the numerical coefficient of each intermediate value modulo 100 if it's positive, and modulo -100 if it's negative: ((+12*x*y+-59*x*y++58*x*y)+(+36*x*y++13*x*y++93*x*y)+(+96*x*y+-55*x*y++73*x*y)). If possible, factor by grouping the final result. A: ((+12*x*y+-59*x*y++58*x*y)+(+36*x*y++13*x*y++93*x*y)+(+96*x*y+-55*x*y++73*x*y))=+67*x*y. Q: Solve the following algebraic expression taking the numerical coefficient of each intermediate value modulo 100 if it's positive, and modulo -100 if it's negative: (-85*a*y+((-98*a*x*y+2*a*x*y)+0*x*y*a)). If possible, factor by grouping the final result.

Symbolic CoT	Q: Solve the following algebraic expression taking the numerical coefficient of each intermediate value modulo 100 if it's positive, and modulo -100 if it's negative: (+39*a*b*y++15*a*b*x*y). If possible, factor by grouping the final result. A: (+39*a*b*y++15*a*b*x*y)= +3*a*b*y*(5*x+13). Q: Solve the following algebraic expression taking the numerical coefficient of each intermediate value modulo 100 if it's positive, and modulo -100 if it's negative: (+21*x*y+(-26*x*y+-92*x*y)). If possible, factor by grouping the final result. A: (+21*x*y+(-26*x*y+-92*x*y))= (+21*x*y+-18*x*y)= +3*x*y. Q: Solve the following algebraic expression taking the numerical coefficient of each intermediate value modulo 100 if it's positive, and modulo -100 if it's negative: ((+45*b*x++22*b*x+-47*b*x)+-62*b*x*y). If possible, factor by grouping the final result. A: ((+45*b*x++22*b*x+-47*b*x)+-62*b*x*y)= (+20*b*x+-62*b*x*y)= -2*b*x*(31*y-10). Q: Solve the following algebraic expression taking the numerical coefficient of each intermediate value modulo 100 if it's positive, and modulo -100 if it's negative: ((-78*b*x*y+(+50*b*x*y+-22*b*x*y))+((-b*x+-57*b*x)+(-38*b*x+-99*b*x))). If possible, factor by grouping the final result.

Verbal CoT	Q: Solve the following algebraic expression taking the numerical coefficient of each intermediate value modulo 100 if it's positive, and modulo -100 if it's negative: (+10*a*b*x*y+-23*a*b*x). If possible, factor by grouping the final result. A: The expression we need to solve is: (+10*a*b*x*y+-23*a*b*x). Simplifying the expression and factoring by grouping, we get to the final result: +a*b*x*(10*y-23) Q: Solve the following algebraic expression taking the numerical coefficient of each intermediate value modulo 100 if it's positive, and modulo -100 if it's negative: (-8*a*x*y+(-38*a*x+-70*a*x)). If possible, factor by grouping the final result. A: Let us recall the expression to be solved: (-8*a*x*y+(-38*a*x+-70*a*x)). By solving a simple expression, we obtain: (-8*a*x*y+-8*a*x). As this expression is in a simple form, we can get to the final result factoring by grouping: -8*a*x*(y+1) Q: Solve the following algebraic expression taking the numerical coefficient of each intermediate value modulo 100 if it's positive, and modulo -100 if it's negative: (+31*a*b*y+(-50*a*b*x+-64*a*b*x+-46*a*b*x)). If possible, factor by grouping the final result. A: We need to solve the following expression: (+31*a*b*y+(-50*a*b*x+-64*a*b*x+-46*a*b*x)). Taking an immediate solution step, we obtain: (+31*a*b*y+-60*a*b*x). Taking an immediate solution step and factoring by grouping, we get to the final result: -a*b*(60*x-31*y). Q: Solve the following algebraic expression taking the numerical coefficient of each intermediate value modulo 100 if it's positive, and modulo -100 if it's negative: ((-78*b*x*y+(+50*b*x*y+-22*b*x*y))+((-b*x+-57*b*x)+(-38*b*x+-99*b*x))). If possible, factor by grouping the final result.
Zero-shot CoT	Q: Simplify the following algebraic expression, computing the modulo 100 of the numerical coefficient of each intermediate value if it's positive, and the modulo -100 if it's negative: (-85*a*y+((-98*a*x*y+2*a*x*y)+0*x*y*a)). If possible, factor by grouping the final result. A: Let's think step-by-step.

Appendix B

Neural Rewriting Systems training details

B.1 Dataset statistics

As described in Section 4.4.2, the development sets for the Neural Rewriting System (NRS) across the four datasets—logic, listops, arithmetic, and algebra—were constructed to capture a diverse range of formula complexities. Each development set is composed of multiple subsets with varying nesting levels, alongside intermediate formulas generated during the resolution process of the main ones. By design, the number of unique formulas available in splits with simpler formulas is smaller than in splits with more complex ones, due to the combinatorial nature of the problem. Despite these differences in the number of unique formulas per split, during training, samples are drawn from each split with equal probability. This ensures balanced exposure across the splits, allowing the model to generalize across different formula complexities. Refer to Tables B.1 and B.2 for the exact number of samples in each development split for the four tasks. We report in Table B.3 the number of (unique) samples in the test sets of the four tasks used across all experiments.

Task	Training set	In-distribution val. set	Out-of-distribution val. set
Logic	238436	710	900
ListOps	840209	2332	900
Arithmetic	399218	180	60
Algebra	323787	900	300

Table B.1: No. of unique samples in the NRS and FastNRS development sets.

B.2 Fast Neural Rewriting System

Selector In all models, we used four attention heads and a hidden state in the feed-forward layers that was four times larger than the embedding size. We trained the models using the Adam optimizer with default parameters, a batch size of 512, a dropout probability of 10%

Task	Training set	In-distribution val. set
Logic	155	45
ListOps	22158	5541
Arithmetic	30510	7628
Algebra	18992	4749

Table B.2: Number of unique samples in the NRS and FastNRS Solver development sets.

Task	Num. samples
Logic	1200
ListOps	600
Arithmetic	600
Algebra	600

Table B.3: Number of unique samples in the test sets

and a cosine annealing schedule of the learning rate with 1000 linear warm-up iterations. We tuned the embedding size, the number of encoder layers, the width of the diagonal window applied to the self-attention matrix and the learning rate using a random search. For all tasks, we searched hyperparameters values in the following ranges: {128, 256, 512} for the embedding size, [1, 9] for the number of encoder layers and [1e-6, 6e-6] for the learning rate. All models were trained using the Adam optimizer for 5,000 iterations, apart from the multi-domain model which was trained for 7,000 iterations. The final values chosen after tuning each hyperparameter are reported in Table B.4.

Solver We tuned the hyperparameters of the Solver using a random search on the embedding size, the number of encoder and decoder layers, the dropout rate, and the learning rate. In all models, we used four attention heads and a hidden state in the feed-forward layers that was four times larger than the embedding size. We trained the models using the Adam optimizer with default parameters, a batch size of 512 and a cosine annealing schedule of the learning rate. The models were trained for 10,000 iterations in the case of Logic and ListOps tasks and for 40,000 and 100,000 iterations in the case of Algebra and Arithmetic tasks, respectively. For all tasks, we searched hyperparameters values in the following ranges: {64, 128, 256} for the embedding size, [1, 4] for the number of encoder and decoder layers, [0.1, 0.4] for the dropout probability, [1e-5, 1e-4] for the learning rate and [1000, 2000] for the number of warmup iterations. The final values chosen after tuning each hyperparameter are reported in Table B.5.

B.2.1 Analysis on Width of the Self-Attention Window and Model Depth

Figures B.1 and B.2 illustrate the impact of both the width of the self-attention diagonal window and the number of layers in the FastNRS Selector model on sequence accuracy, measured on an out-of-distribution set of samples during training. We report the mean and standard deviation of a group of three runs with different random seeds. As shown in the plots, there is a clear inverse relationship between the width of the self-attention window

	Logic	ListOps	Arithmetic	Algebra	Multi-domain
Embedding size	256	256	256	256	256
Num. Enc. Layers	3	4	4	6	4
Width self-attn window	1	1	1	1	1
Learning rate	3.55e-05	3.65e-05	2.66e-05	4.49e-05	1.69e-05

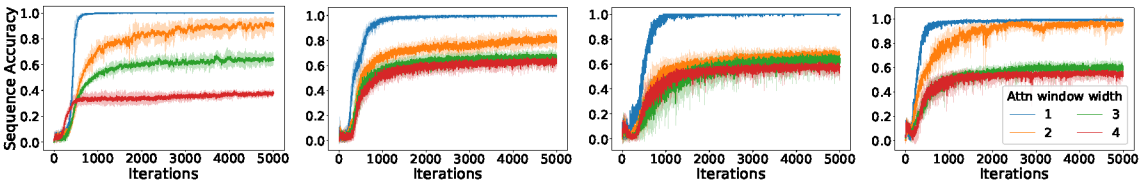
Table B.4: FastNRS Selector tuned hyperparameters values for each test scenario.

	Logic	ListOps	Arithmetic	Algebra	Multi-domain
Embedding size	64	128	256	256	320
Num. Enc. Layers	1	2	3	2	4
Num. Dec. Layers	1	2	3	2	4
Dropout	0.18	0.18	0.1	0.33	0.13
Learning rate	9.23e-05	9.59e-05	9e-05	8e-05	6.19e-05
Warm-up it.	1282	1910	1500	1500	1714

Table B.5: FastNRS Solver tuned hyperparameters values for each test scenario.

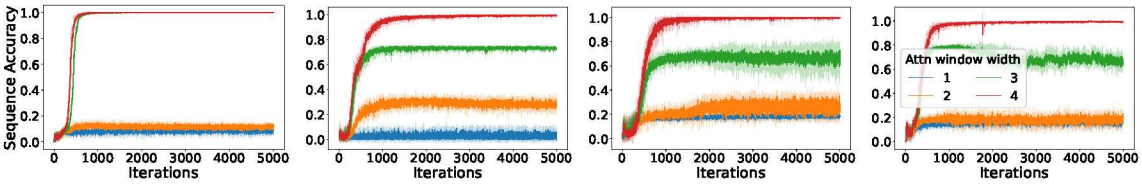
and model performance, consistent across all the domains we consider. Specifically, as the window width increases, the model becomes progressively less capable of generalizing on out-of-distribution samples. Peak performance, indicated by the highest accuracy, is achieved when the window width is set to 1, suggesting that a narrower focus in self-attention enhances the model's ability to generalize to out-of-distribution data.

In contrast, there is a direct relationship between the number of layers and model performance: as the number of layers increases, the model's generalization ability improves, leading to better accuracy on both in-distribution and out-of-distribution samples. We also notice that all the models with different window widths were equally able to fit the training set with no sign of overfitting on the in-distribution validation set. On the contrary, models with varying numbers of layers showed similar performance across the training, in-distribution, and out-of-distribution validation sets (not shown). Both analyses use model hyperparameters that were selected through hyperparameter tuning and are consistent with those applied in the rest of the experiments.



(a) Logic (b) Listops (c) Arithmetic (d) Algebra

Figure B.1: Impact of self-attention window width on out-of-distribution sequence accuracy.



(a) Logic (b) Listops (c) Arithmetic (d) Algebra

Figure B.2: Impact of number of layers on out-of-distribution sequence accuracy.

B.3 Neural Rewriting System

Selector In all models we used four attention heads and a hidden state in the feed-forward layers that was four times larger than the embedding size. Selector models were trained for 20,000 iterations for the Logic and ListOps tasks, and 30,000 iterations for the Arithmetic and Algebra tasks. We trained the models using the Adam optimizer with default parameters, a batch size of 512 (256 for Algebra) and a cosine annealing schedule of the learning rate with warm-up. We tuned the embedding size, the number of encoder and decoder layers, the width of the diagonal window applied to the self-attention matrix, the dropout rate, the learning rate, the number of warm-up iterations and the value of gain parameter for initialization of the self-attention layers using a random search. For all tasks, we searched hyperparameters values in the following ranges: {128, 256, 512} for the embedding size, [1, 3] for the width of the diagonal self-attention window, [2, 5] for the number of encoder and decoder layers, [0.1, 0.4] for the dropout probability, [1e-5, 6e-5] for the learning rate, [500, 3000] for the number of warm-up iterations, [0.5, 2.5] for the initialization gain parameter. The final values chosen after tuning each hyperparameter are reported in Table B.6.

Solver In our experiments we used the same Solver modules both in the FastNRS and in the NRS, thus the hyperparameters for the NRS Solver correspond to those detailed for the FastNRS Solver in Table B.5.

	Logic	ListOps	Arithmetic	Algebra	Multi-domain
Embedding size	256	256	256	256	256
Width self-attn window	2	2	3	3	2
Num. Enc. Layers	1	1	3	4	5
Num. Dec. Layers	2	2	2	2	2
Dropout	0.29	0.37	0.17	0.20	0.10
Learning rate	2.7e-5	2.65e-5	2.35e-5	5.54e-5	7.86e-5
Warm-up it.	1600	1700	1900	2900	1500
MHA init. gain	0.97	0.71	1.69	0.75	1.00

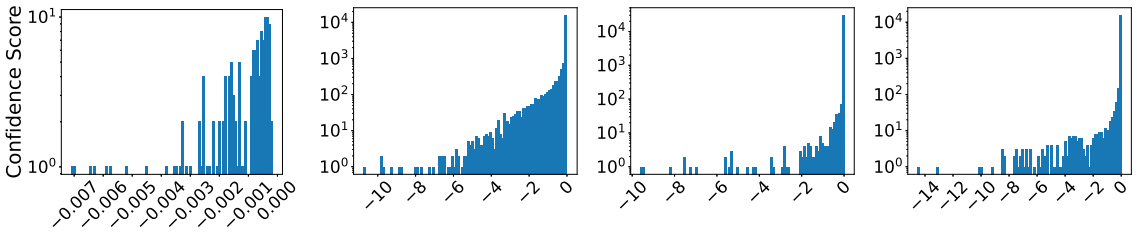
Table B.6: NRS Selector tuned hyperparameters values for each test scenario.

Appendix C

Confidence scores in Neural Rewriting Systems

The inference-time behavior of both the FastNRS and the NRS can be regulated by the value of some hyperparameters, as explained in Sections 4.3.1 and 4.3.2. In both cases, the values of these hyperparameters can be meaningfully chosen taking into account the distribution of the confidence of some module in the architecture (the Selector, in the case of the NRS, and the Solver, in the case of the FastNRS). To justify our choice of hyperparameters and give more context to the reader, we provide here a visual representation of the distribution of these values.

C.1 Distribution of FastNRS Solver confidence scores



(a) Logic (b) Listops (c) Arithmetic (d) Algebra

Figure C.1: Distribution of Solver confidence scores on training samples (y-axis in log scale).

The conditional replacement of leaf formulas in the FastNRS operated by the `cond_repl` function is regulated by a threshold on Solver confidence. For both the single- and multi-domain versions of the FastNRS, we determined these thresholds based on the distribution of these scores on training samples, as mentioned in Section 4.3.2. Specifically, the thresholds used were -6 for ListOps, -2 for Arithmetic, -3 for Algebra, and -0.005 for Logic. The distributions of Solver confidence scores on training samples, which informed these thresholds, are represented in Figure C.1. The plots can provide insight into how frequently high-confidence predictions occur.

C.2 NRS Selector confidence scores

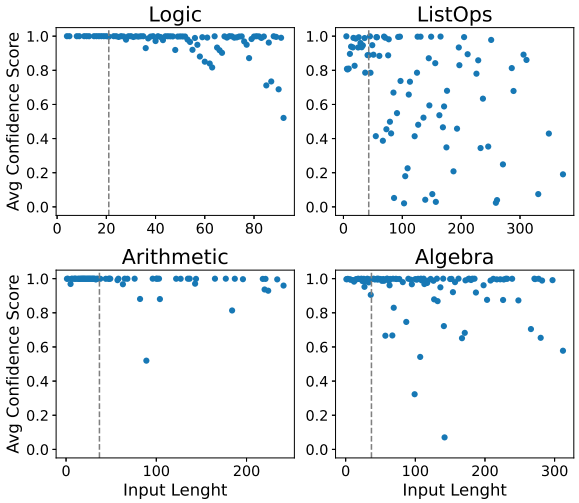


Figure C.2: Average single-domain NRS Selector confidence scores by input length. The vertical line represents the maximum length of training formulas.

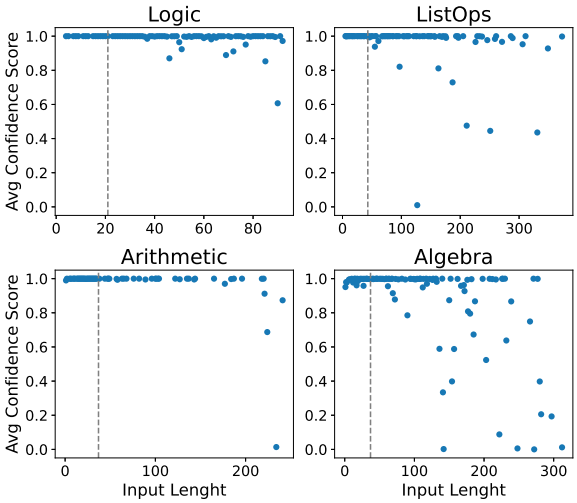


Figure C.3: Average multi-domain NRS Selector confidence scores by input length. The vertical line represents the maximum length of training formulas.

As described in Section 4.3.1, the Dynamic Windowing mechanism in the NRS Selector is regulated by a threshold T, which is used to determine on which formulas the mechanism should be applied. We select these thresholds by examining the average Selector confidence score for inputs of the same length, and choose the value corresponding to a decrease in average Selector confidence. We measure the average Selector confidence on several formulas of different lengths and nesting levels drawn from both the in-distribution and out-of-distribution validation sets. The distribution of these values is represented in Figures C.2 and C.3.

Appendix D

Neural Data Router training details

	Logic	ListOps	Arithmetic	Algebra
Num. Enc. Layers	19	5	11	17
Embedding size	256	512	512	512
Attention heads	8	16	8	16
FF size	1024	1024	2048	2048
Learning rate	2.33e-04	4.13e-04	7.64e-04	9.59e-04
Dropout	0.45	0.09	0.05	0.40
Attention dropout	0.38	0.49	0.06	0.18
Weight decay	0.02	0.09	0.08	0.03

Table D.1: NDR tuned hyperparameters values for each task.

Here we report the hyperparameters of the best Neural Data Router configuration we selected for each problem. We searched hyperparameters values in the same ranges used in the original paper. Models were trained using the AdamW optimizer for 5,000 iterations in the case of Logic, 30,000 iterations in the case of ListOps, and 100,000 iterations in the case of Arithmetic and Algebra. We used a batch size of 512 for all task except Algebra, for which it was 256. The final values chosen after tuning are reported in Table D.1.

References

[1] S. Russell and P. Norvig. *Artificial intelligence*. en. 4th ed. Upper Saddle River, NJ: Pearson, Nov. 2020.

[2] C. E. Shannon. "XXII. Programming a computer for playing chess". In: *Lond. Edinb. Dublin Philos. Mag. J. Sci.* 41.314 (Mar. 1950), pp. 256–275.

[3] U. Güçlü and M. A. J. van Gerven. "Deep neural networks reveal a gradient in the complexity of neural representations across the ventral stream". en. In: *J. Neurosci.* 35.27 (July 2015), pp. 10005–10014.

[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning representations by back-propagating errors". en. In: *Nature* 323.6088 (Oct. 1986), pp. 533–536.

[5] S. Löwel and W. Singer. "Selection of intrinsic horizontal connections in the visual cortex by correlated neuronal activity". en. In: *Science* 255.5041 (Jan. 1992), pp. 209–212.

[6] D. O. Hebb. *The organization of behavior*. London, England: Psychology Press, Apr. 2005.

[7] J. Devlin, M. Chang, K. Lee, and K. Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Ed. by J. Burstein, C. Doran, and T. Solorio. Association for Computational Linguistics, 2019, pp. 4171–4186. DOI: 10.18653/V1/N19-1423. URL: https://doi.org/10.18653/v1/n19-1423.

[8] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[9] A. Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL: https://openreview.net/forum?id=YicbFdNTTy.

[10] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever. *Robust Speech Recognition via Large-Scale Weak Supervision*. 2022. arXiv: 2212.04356 [eess.AS]. URL: https://arxiv.org/abs/2212.04356.

[11] J. Alayrac et al. “Flamingo: a Visual Language Model for Few-Shot Learning”. In: *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. 2022. URL: http://papers.nips.cc/paper%5C_files/paper/2022/hash/960a172bc7fbf0177ccccbb411a7d800-Abstract-Conference.html.

[12] H. Liu, C. Li, Q. Wu, and Y. J. Lee. “Visual Instruction Tuning”. In: *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*. Ed. by A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine. 2023. URL: http://papers.nips.cc/paper%5C_files/paper/2023/hash/6dcf277ea32ce3288914faf369fe6de0-Abstract-Conference.html.

[13] T. B. Brown et al. “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*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin. 2020.

[14] D. Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, Oct. 2011.

[15] C. R. Jones and B. K. Bergen. *Does GPT-4 pass the Turing test?* 2024. arXiv: 2310.20216 [cs.AI]. URL: https://arxiv.org/abs/2310.20216.

[16] D. Jannai, A. Meron, B. Lenz, Y. Levine, and Y. Shoham. *Human or Not? A Gamified Approach to the Turing Test*. 2023. arXiv: 2305.20010 [cs.AI]. URL: https://arxiv.org/abs/2305.20010.

[17] S. Lee, W. Sim, D. Shin, S. Hwang, W. Seo, J. Park, S. Lee, S. Kim, and S. Kim. *Reasoning Abilities of Large Language Models: In-Depth Analysis on the Abstraction and Reasoning Corpus*. 2024. arXiv: 2403.11793 [cs.CL]. URL: https://arxiv.org/abs/2403.11793.

[18] A. Srivastava et al. “Beyond the Imitation Game: Quantifying and extrapolating the capabilities of language models”. In: *Transactions on Machine Learning Research* (2023). ISSN: 2835-8856. URL: https://openreview.net/forum?id=uyTL5Bvosj.

[19] I. Dumontheil. “Development of abstract thinking during childhood and adolescence: the role of rostrolateral prefrontal cortex”. en. In: *Dev. Cogn. Neurosci.* 10 (Oct. 2014), pp. 57–76.

[20] D. E. Nee, A. Jahn, and J. W. Brown. “Prefrontal cortex organization: dissociating effects of temporal abstraction, relational abstraction, and integration with FMRI”. en. In: *Cereb. Cortex* 24.9 (Sept. 2014), pp. 2377–2387.

[21] D. Silver et al. “Mastering the game of Go without human knowledge”. en. In: *Nature* 550.7676 (Oct. 2017), pp. 354–359.

[22] J. A. Fodor and Z. W. Pylyshyn. “Connectionism and cognitive architecture: a critical analysis”. en. In: *Cognition* 28.1-2 (Mar. 1988), pp. 3–71.

[23] J. A. Fodor. *The language of thought*. The Language and Thought Series. London, England: Harvard University Press, July 1979.

[24] T. Mikolov, A. Joulin, and M. Baroni. "A Roadmap Towards Machine Intelligence". In: *Computational Linguistics and Intelligent Text Processing*. Lecture notes in computer science. Cham: Springer International Publishing, 2018, pp. 29–61.

[25] A. Santoro, A. Lampinen, K. Mathewson, T. Lillicrap, and D. Raposo. *Symbolic Behaviour in Artificial Intelligence*. 2022. arXiv: 2102.03406 [cs.AI]. URL: https://arxiv.org/abs/2102.03406.

[26] C. Burns et al. "Weak-to-Strong Generalization: Eliciting Strong Capabilities With Weak Supervision". In: *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL: https://openreview.net/forum?id=ghNRg2mEgN.

[27] N. Jones. "OpenAI's chief scientist helped to create ChatGPT - while worrying about AI safety". en. In: *Nature* 624.7992 (Dec. 2023), p. 503.

[28] H. Murphy and C. Criddle. "Meta AI chief says large language models will not reach human intelligence". In: *Financial Times* (May 22, 2024). URL: https://www.ft.com/content/23fab126-f1d3-4add-a457-207a25730ad9?trk=public_post_comment-text (visited on 03/09/2024).

[29] Y. LeCun. "A Path Towards Autonomous Machine Intelligence Version 0.9.2, 2022-06-27". https://api.semanticscholar.org/CorpusID:251881108. 2022.

[30] A. Zador et al. "Catalyzing next-generation Artificial Intelligence through NeuroAI". en. In: *Nat. Commun.* 14.1 (Mar. 2023), p. 1597.

[31] P. Hitzler. *Neuro-Symbolic Artificial Intelligence: The state of the art*. en. Amsterdam, NY: IOS Press, Jan. 2022.

[32] G. Marcus. "The Next Decade in AI: Four Steps Towards Robust Artificial Intelligence". In: *CoRR* abs/2002.06177 (2020). arXiv: 2002.06177. URL: https://arxiv.org/abs/2002.06177.

[33] A. D. Garcez and L. C. Lamb. "Neurosymbolic AI: The 3rd wave". en. In: *Artif. Intell. Rev.* 56.11 (Nov. 2023), pp. 12387–12406.

[34] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, et al. "Emergent abilities of large language models". In: *arXiv preprint arXiv:2206.07682* (2022).

[35] F. Petruzzellis, A. Testolin, and A. Sperduti. "Assessing the Emergent Symbolic Reasoning Abilities of Llama Large Language Models". In: *Artificial Neural Networks and Machine Learning – ICANN 2024*. Ed. by M. Wand, K. Malinovská, J. Schmidhuber, and I. V. Tetko. Cham: Springer Nature Switzerland, 2024, pp. 266–276. ISBN: 978-3-031-72344-5.

[36] F. Petruzzellis, A. Testolin, and A. Sperduti. "Benchmarking GPT-4 on Algorithmic Problems: A Systematic Evaluation of Prompting Strategies". In: *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*. Ed. by N. Calzolari, M.-Y. Kan, V. Hoste, A. Lenci, S. Sakti, and N. Xue. Torino, Italia: ELRA and ICCL, May 2024, pp. 2161–2177. URL: https://aclanthology.org/2024.lrec-main.195.

[37] F. Petruzzellis, A. Testolin, and A. Sperduti. "A Hybrid System for Systematic Generalization in Simple Arithmetic Problems". In: *Proceedings of the 17th International Workshop on Neural-Symbolic Learning and Reasoning*. 2023.

[38] F. Petruzzellis, A. Testolin, and A. Sperduti. "A Neural Rewriting System to Solve Algorithmic Problems". To appear in Proceedings of the 27th European Conference on Artificial Intelligence (ECAI 24). 2024.

[39] N. Chomsky. "Three models for the description of language". In: *IRE Transactions on Information Theory* 2.3 (1956), pp. 113–124. DOI: 10.1109/TIT.1956.1056813.

[40] N. Chomsky. *Aspects of the theory of syntax*. London, England: MIT Press, Jan. 1965.

[41] Fahlman and Hinton. "Connectionist Architectures for Artificial Intelligence". In: *Computer* 20.1 (1987), pp. 100–109. DOI: 10.1109/MC.1987.1663364.

[42] D. E. Rumelhart, J. L. McClelland, and The PDP Research Group. *Parallel distributed processing*. Parallel Distributed Processing. London, England: MIT Press, Jan. 1986.

[43] D. S. Touretzky. "BoltzCONS: Dynamic Symbol Structures in a Connectionist Network". In: *Artif. Intell.* 46.1-2 (1990), pp. 5–46. DOI: 10.1016/0004-3702(90)90003-I. URL: https://doi.org/10.1016/0004-3702(90)90003-I.

[44] G. E. Hinton. "Mapping Part-Whole Hierarchies into Connectionist Networks". In: *Artif. Intell.* 46.1-2 (1990), pp. 47–75. DOI: 10.1016/0004-3702(90)90004-J. URL: https://doi.org/10.1016/0004-3702(90)90004-J.

[45] P. Smolensky. "Tensor Product Variable Binding and the Representation of Symbolic Structures in Connectionist Systems". In: *Artif. Intell.* 46.1-2 (1990), pp. 159–216. DOI: 10.1016/0004-3702(90)90007-M. URL: https://doi.org/10.1016/0004-3702(90)90007-M.

[46] D. Hupkes, V. Dankers, M. Mul, and E. Bruni. "Compositionality Decomposed: How do Neural Networks Generalise?" In: *J. Artif. Intell. Res.* 67 (2020), pp. 757–795. DOI: 10.1613/jair.1.11674. URL: https://doi.org/10.1613/jair.1.11674.

[47] B. M. Lake and M. Baroni. "Generalization without Systematicity: On the Compositional Skills of Sequence-to-Sequence Recurrent Networks". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by J. G. Dy and A. Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 2879–2888. URL: http://proceedings.mlr.press/v80/lake18a.html.

[48] N. Kim and T. Linzen. "COGS: A Compositional Generalization Challenge Based on Semantic Interpretation". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*. Ed. by B. Webber, T. Cohn, Y. He, and Y. Liu. Association for Computational Linguistics, 2020, pp. 9087–9105. DOI: 10.18653/v1/2020.emnlp-main.731. URL: https://doi.org/10.18653/v1/2020.emnlp-main.731.

[49] A. Liska, G. Kruszewski, and M. Baroni. "Memorize or generalize? Searching for a compositional RNN in a haystack". In: *CoRR* abs/1802.06467 (2018). arXiv: 1802.06467. URL: http://arxiv.org/abs/1802.06467.

[50] D. Keysers et al. "Measuring Compositional Generalization: A Comprehensive Method on Realistic Data". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=SygcCnNKwr.

[51] R. Csordás, K. Irie, and J. Schmidhuber. "CTL++: Evaluating Generalization on Never-Seen Compositional Patterns of Known Functions, and Compatibility of Neural Representations". In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*. Ed. by Y. Goldberg, Z. Kozareva, and Y. Zhang. Association for Computational Linguistics, 2022, pp. 9758–9767. DOI: 10.18653/v1/2022.emnlp-main.662. URL: https://doi.org/10.18653/v1/2022.emnlp-main.662.

[52] 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*. Ed. by I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett. 2017, pp. 5998–6008. URL: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[53] D. Bahdanau, K. Cho, and Y. Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. 2015. URL: http://arxiv.org/abs/1409.0473.

[54] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma. "Linformer: Self-Attention with Linear Complexity". In: *CoRR* abs/2006.04768 (2020). arXiv: 2006.04768. URL: https://arxiv.org/abs/2006.04768.

[55] I. Beltagy, M. E. Peters, and A. Cohan. "Longformer: The Long-Document Transformer". In: *CoRR* abs/2004.05150 (2020). arXiv: 2004.05150. URL: https://arxiv.org/abs/2004.05150.

[56] J. Su et al. "RoFormer: Enhanced transformer with Rotary Position Embedding". In: *Neurocomputing* 568 (2024), p. 127063. DOI: 10.1016/J.NEUCOM.2023.127063. URL: https://doi.org/10.1016/j.neucom.2023.127063.

[57] O. Press, N. A. Smith, and M. Lewis. "Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation". In: *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL: https://openreview.net/forum?id=R8sQPpGCv0.

[58] A. Ruoss, G. Delétang, T. Genewein, J. Grau-Moya, R. Csordás, M. Bennani, S. Legg, and J. Veness. "Randomized Positional Encodings Boost Length Generalization of Transformers". In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*. Ed. by A. Rogers, J. L. Boyd-Graber, and N. Okazaki. Association for Computational Linguistics, 2023, pp. 1889–1903. DOI: 10.18653/v1/2023.acl-short.161. URL: https://doi.org/10.18653/v1/2023.acl-short.161.

[59] Y. Li and J. L. McClelland. "Representations and Computations in Transformers that Support Generalization on Structured Tasks". In: *Trans. Mach. Learn. Res.* 2023 (2023). URL: https://openreview.net/forum?id=oFC2LAqS6Z.

[60] B. Newman, J. Hewitt, P. Liang, and C. D. Manning. "The EOS Decision and Length Extrapolation". In: *Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*. Ed. by A. Alishahi, Y. Belinkov, G. Chrupała, D. Hupkes, Y. Pinter, and H. Sajjad. Online: Association for Computational Linguistics, Nov. 2020, pp. 276–291. DOI: 10.18653/v1/2020.blackboxnlp-1.26. URL: https://aclanthology.org/2020.blackboxnlp-1.26.

[61] R. Csordás, K. Irie, and J. Schmidhuber. "The Devil is in the Detail: Simple Tricks Improve Systematic Generalization of Transformers". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*. Ed. by M. Moens, X. Huang, L. Specia, and S. W. Yih. Association for Computational Linguistics, 2021, pp. 619–634. DOI: 10.18653/v1/2021.emnlp-main.49. URL: https://doi.org/10.18653/v1/2021.emnlp-main.49.

[62] M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and L. Kaiser. "Universal Transformers". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=HyzdRiR9Y7.

[63] Y. Li and J. L. McClelland. "Systematic Generalization and Emergent Structures in Transformers Trained on Structured Tasks". In: *CoRR* abs/2210.00400 (2022). DOI: 10.48550/arXiv.2210.00400. arXiv: 2210.00400. URL: https://doi.org/10.48550/arXiv.2210.00400.

[64] P. Shaw, J. Uszkoreit, and A. Vaswani. "Self-Attention with Relative Position Representations". 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)*. Ed. by M. A. Walker, H. Ji, and A. Stent. Association for Computational Linguistics, 2018, pp. 464–468. DOI: 10.18653/V1/N18-2074. URL: https://doi.org/10.18653/v1/n18-2074.

[65] A. J. Nam, M. Abdool, T. Maxfield, and J. L. McClelland. "Out-of-Distribution Generalization in Algorithmic Reasoning Through Curriculum Learning". In: *CoRR* abs/2210.03275 (2022). DOI: 10.48550/arXiv.2210.03275. arXiv: 2210.03275. URL: https://doi.org/10.48550/arXiv.2210.03275.

[66] B. M. Lake and M. Baroni. "Human-like systematic generalization through a meta-learning neural network". In: *Nat.* 623.7985 (2023), pp. 115–121. DOI: 10.1038/S41586-023-06668-3. URL: https://doi.org/10.1038/s41586-023-06668-3.

[67] S. Ontañón, J. Ainslie, Z. Fisher, and V. Cvicek. "Making Transformers Solve Compositional Tasks". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*. Ed. by S. Muresan, P. Nakov, and A. Villavicencio. Association for Com-

putational Linguistics, 2022, pp. 3591–3607. DOI: 10.18653/v1/2022.acl-long.251. URL: https://doi.org/10.18653/v1/2022.acl-long.251.

[68] M. Setzler, S. Howland, and L. A. Phillips. "Recursive Decoding: A Situated Cognition Approach to Compositional Generation in Grounded Language Understanding". In: *CoRR* abs/2201.11766 (2022). arXiv: 2201.11766. URL: https://arxiv.org/abs/2201.11766.

[69] R. Deshpande, J. Chen, and I. G. Lee. "RecT: A Recursive Transformer Architecture for Generalizable Mathematical Reasoning". In: *International Workshop on Neural-Symbolic Learning and Reasoning*. 2021.

[70] R. Csordás, K. Irie, and J. Schmidhuber. "The Neural Data Router: Adaptive Control Flow in Transformers Improves Systematic Generalization". In: *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL: https://openreview.net/forum?id=KBQP4A%5C_J1K.

[71] N. Nangia and S. R. Bowman. "ListOps: A Diagnostic Dataset for Latent Tree Learning". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 2-4, 2018, Student Research Workshop*. Ed. by S. R. Cordeiro, S. Oraby, U. Pavalanathan, and K. Rim. Association for Computational Linguistics, 2018, pp. 92–99. DOI: 10.18653/v1/n18-4013. URL: https://doi.org/10.18653/v1/n18-4013.

[72] M. Nye, A. Solar-Lezama, J. Tenenbaum, and B. M. Lake. "Learning Compositional Rules via Neural Program Synthesis". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 10832–10842. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/7a685d9edd95508471a9d3d6fcace432-Paper.pdf.

[73] Q. Liu, S. An, J.-G. Lou, B. Chen, Z. Lin, Y. Gao, B. Zhou, N. Zheng, and D. Zhang. "Compositional Generalization by Learning Analytical Expressions". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 11416–11427. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/83adc9225e4deb67d7ce42d58fe5157c-Paper.pdf.

[74] X. Chen, C. Liang, A. W. Yu, D. Song, and D. Zhou. "Compositional Generalization via Neural-Symbolic Stack Machines". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 1690–1701. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/12b1e42dc0746f22cf361267de07073f-Paper.pdf.

[75] D. Saxton, E. Grefenstette, F. Hill, and P. Kohli. "Analysing Mathematical Reasoning Abilities of Neural Models". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=H1gR5iR5FX.

[76] A. Graves, G. Wayne, and I. Danihelka. "Neural Turing Machines". In: *CoRR* abs/1410.5401 (2014). arXiv: 1410.5401. URL: http://arxiv.org/abs/1410.5401.

[77] A. Graves et al. “Hybrid computing using a neural network with dynamic external memory”. In: *Nat.* 538.7626 (2016), pp. 471–476. DOI: 10.1038/NATURE20101. URL: https://doi.org/10.1038/nature20101.

[78] Ç. Gülçehre, S. Chandar, K. Cho, and Y. Bengio. “Dynamic Neural Turing Machine with Continuous and Discrete Addressing Schemes”. In: *Neural Comput.* 30.4 (2018). DOI: 10.1162/NECO_A_01060. URL: https://doi.org/10.1162/neco%5C_a%5C_01060.

[79] R. Csordás and J. Schmidhuber. “Improving Differentiable Neural Computers Through Memory Masking, De-allocation, and Link Distribution Sharpness Control”. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=HyGEM3C9KQ.

[80] W. Zaremba and I. Sutskever. “Reinforcement Learning Neural Turing Machines”. In: *CoRR* abs/1505.00521 (2015). arXiv: 1505.00521. URL: http://arxiv.org/abs/1505.00521.

[81] J. Weston, S. Chopra, and A. Bordes. “Memory Networks”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. 2015. URL: http://arxiv.org/abs/1410.3916.

[82] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus. “End-To-End Memory Networks”. In: *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett. 2015, pp. 2440–2448. URL: https://proceedings.neurips.cc/paper/2015/hash/8fb21ee7a2207526da55a679f0332de2-Abstract.html.

[83] Y. Wu, G. Wayne, A. Graves, and T. P. Lillicrap. “The Kanerva Machine: A Generative Distributed Memory”. In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL: https://openreview.net/forum?id=S1HlA-ZAZ.

[84] J. W. Rae, J. J. Hunt, I. Danihelka, T. Harley, A. W. Senior, G. Wayne, A. Graves, and T. Lillicrap. “Scaling Memory-Augmented Neural Networks with Sparse Reads and Writes”. In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. Ed. by D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett. 2016, pp. 3621–3629. URL: https://proceedings.neurips.cc/paper/2016/hash/3fab5890d8113d0b5a4178201dc842ad-Abstract.html.

[85] G. Lample and F. Charton. “Deep Learning for Symbolic Mathematics”. In: *ArXiv* abs/1912.01412 (2019).

[86] L. Kaiser and I. Sutskever. "Neural GPUs Learn Algorithms". In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. 2016. URL: http://arxiv.org/abs/1511.08228.

[87] 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*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett. 2015, pp. 2692–2700.

[88] A. Trask, F. Hill, S. E. Reed, J. Rae, C. Dyer, and P. Blunsom. "Neural Arithmetic Logic Units". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/0e64a7b00c83e3d22ce6b3acf2c582b6-Paper.pdf.

[89] A. Madsen and A. R. Johansen. "Neural Arithmetic Units". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=H1gNOeHKPS.

[90] P. Velickovic and C. Blundell. "Neural algorithmic reasoning". In: *Patterns* 2.7 (2021), p. 100273. DOI: 10.1016/J.PATTER.2021.100273. URL: https://doi.org/10.1016/j.patter.2021.100273.

[91] P. Velickovic, A. P. Badia, D. Budden, R. Pascanu, A. Banino, M. Dashevskiy, R. Hadsell, and C. Blundell. "The CLRS Algorithmic Reasoning Benchmark". In: *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*. Ed. by K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato. Vol. 162. Proceedings of Machine Learning Research. PMLR, 2022, pp. 22084–22102. URL: https://proceedings.mlr.press/v162/velickovic22a.html.

[92] K. Xu, J. Li, M. Zhang, S. S. Du, K. Kawarabayashi, and S. Jegelka. "What Can Neural Networks Reason About?" In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=rJxbJeHFPS.

[93] A. J. Dudzik and P. Velickovic. "Graph Neural Networks are Dynamic Programmers". In: *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. 2022. URL: http://papers.nips.cc/paper%5C_files/paper/2022/hash/8248b1ded388fcdbbd121bcdfea3068c-Abstract-Conference.html.

[94] M. M. Bronstein, J. Bruna, T. Cohen, and P. Velickovic. "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges". In: *CoRR* abs/2104.13478 (2021). arXiv: 2104.13478. URL: https://arxiv.org/abs/2104.13478.

[95] K. Xu and P. Veličković. *Recurrent Aggregators in Neural Algorithmic Reasoning*. 2024. arXiv: 2409.07154 [cs.LG]. URL: https://arxiv.org/abs/2409.07154.

[96] W. Bounsi, B. Ibarz, A. Dudzik, J. B. Hamrick, L. Markeeva, A. Vitvitskyi, R. Pascanu, and P. Velickovic. "Transformers meet Neural Algorithmic Reasoners". In: *CoRR* abs/2406.09308 (2024). DOI: 10.48550/ARXIV.2406.09308. arXiv: 2406.09308. URL: https://doi.org/10.48550/arXiv.2406.09308.

[97] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. "Improving language understanding by generative pre-training". 2018.

[98] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. "Language Models are Unsupervised Multitask Learners". 2019.

[99] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. "Scaling Laws for Neural Language Models". In: *CoRR* abs/2001.08361 (2020). arXiv: 2001.08361. URL: https://arxiv.org/abs/2001.08361.

[100] L. Ouyang et al. "Training language models to follow instructions with human feedback". In: *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. 2022. URL: http://papers.nips.cc/paper%5C_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html.

[101] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi. "WinoGrande: an adversarial winograd schema challenge at scale". In: *Commun. ACM* 64.9 (2021), pp. 99–106. DOI: 10.1145/3474381. URL: https://doi.org/10.1145/3474381.

[102] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. "HellaSwag: Can a Machine Really Finish Your Sentence?" In: *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. Ed. by A. Korhonen, D. R. Traum, and L. Màrquez. Association for Computational Linguistics, 2019, pp. 4791–4800. DOI: 10.18653/V1/P19-1472. URL: https://doi.org/10.18653/v1/p19-1472.

[103] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. "Measuring Massive Multitask Language Understanding". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL: https://openreview.net/forum?id=d7KBjmI3GmQ.

[104] L. Gao et al. "The Pile: An 800GB Dataset of Diverse Text for Language Modeling". In: *CoRR* abs/2101.00027 (2021). arXiv: 2101.00027. URL: https://arxiv.org/abs/2101.00027.

[105] J. W. Rae et al. "Scaling Language Models: Methods, Analysis & Insights from Training Gopher". In: *CoRR* abs/2112.11446 (2021). arXiv: 2112.11446. URL: https://arxiv.org/abs/2112.11446.

[106] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models". In: *NeurIPS*. 2022. URL: http://papers.nips.cc/paper%5C_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

[107] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. "Large Language Models are Zero-Shot Reasoners". In: *NeurIPS*. 2022. URL: http://papers.nips.cc/paper%5C_files/paper/2022/hash/8bb0d291acd4acf06ef112099c16f326-Abstract-Conference.html.

[108] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan. "Tree of Thoughts: Deliberate Problem Solving with Large Language Models". In: *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*. Ed. by A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine. 2023. URL: http://papers.nips.cc/paper%5C_files/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract-Conference.html.

[109] X. Wang et al. "Self-Consistency Improves Chain of Thought Reasoning in Language Models". In: *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. 2023.

[110] O. Press, M. Zhang, S. Min, L. Schmidt, N. Smith, and M. Lewis. "Measuring and Narrowing the Compositionality Gap in Language Models". In: *Findings of the Association for Computational Linguistics: EMNLP 2023*. Ed. by H. Bouamor, J. Pino, and K. Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 5687–5711. DOI: 10.18653/v1/2023.findings-emnlp.378. URL: https://aclanthology.org/2023.findings-emnlp.378.

[111] M. Zhang, J. He, S. Lei, M. Yue, L. Wang, and C. Lu. "Can LLM Find the Green Circle? Investigation and Human-Guided Tool Manipulation for Compositional Generalization". In: *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2024, Seoul, Republic of Korea, April 14-19, 2024*. IEEE, 2024, pp. 11996–12000. DOI: 10.1109/ICASSP48485.2024.10446355. URL: https://doi.org/10.1109/ICASSP48485.2024.10446355.

[112] S. Wang, Z. Wei, Y. Choi, and X. Ren. "Can LLMs Reason with Rules? Logic Scaffolding for Stress-Testing and Improving LLMs". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*. Ed. by L. Ku, A. Martins, and V. Srikumar. Association for Computational Linguistics, 2024, pp. 7523–7543. URL: https://aclanthology.org/2024.acl-long.406.

[113] Z. Li, G. Jiang, H. Xie, L. Song, D. Lian, and Y. Wei. "Understanding and Patching Compositional Reasoning in LLMs". In: *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*. Ed. by L. Ku, A. Martins, and V. Srikumar. Association for Computational Linguistics, 2024, pp. 9668–9688. URL: https://aclanthology.org/2024.findings-acl.576.

[114] R. Dhar and A. Søgaard. *From Words to Worlds: Compositionality for Cognitive Architectures*. 2024. arXiv: 2407.13419 [cs.CL]. URL: https://arxiv.org/abs/2407.13419.

[115] N. Dziri et al. "Faith and Fate: Limits of Transformers on Compositionality". In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023.

[116] J. Chen, X. Pan, D. Yu, K. Song, X. Wang, D. Yu, and J. Chen. "Skills-in-Context Prompting: Unlocking Compositionality in Large Language Models". In: *CoRR* abs/2308.00304 (2023). DOI: 10.48550/ARXIV.2308.00304. arXiv: 2308.00304. URL: https://doi.org/10.48550/arXiv.2308.00304.

[117] H. Zhou, A. Bradley, E. Littwin, N. Razin, O. Saremi, J. M. Susskind, S. Bengio, and P. Nakkiran. "What Algorithms can Transformers Learn? A Study in Length Generalization". In: *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL: https://openreview.net/forum?id=AssIuHnmHX.

[118] G. Weiss, Y. Goldberg, and E. Yahav. "Thinking Like Transformers". In: *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*. Ed. by M. Meila and T. Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 11080–11090. URL: http://proceedings.mlr.press/v139/weiss21a.html.

[119] C. Anil, Y. Wu, A. Andreassen, A. Lewkowycz, V. Misra, V. V. Ramasesh, A. Slone, G. Gur-Ari, E. Dyer, and B. Neyshabur. "Exploring Length Generalization in Large Language Models". In: *NeurIPS*. 2022. URL: http://papers.nips.cc/paper%5C_files/paper/2022/hash/fb7451e43f9c1c35b774bcfad7a5714b-Abstract-Conference.html.

[120] A. Saparov, R. Y. Pang, V. Padmakumar, N. Joshi, M. Kazemi, N. Kim, and H. He. "Testing the General Deductive Reasoning Capacity of Large Language Models Using OOD Examples". In: *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*. Ed. by A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine. 2023. URL: http://papers.nips.cc/paper%5C_files/paper/2023/hash/09425891e393e64b0535194a81ba15b7-Abstract-Conference.html.

[121] D. Zhou et al. "Least-to-Most Prompting Enables Complex Reasoning in Large Language Models". In: *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL: https://openreview.net/pdf?id=WZH7099tgfM.

[122] M. I. Nye et al. "Show Your Work: Scratchpads for Intermediate Computation with Language Models". In: *CoRR* abs/2112.00114 (2021). arXiv: 2112.00114. URL: https://arxiv.org/abs/2112.00114.

[123] J. Lanchantin, S. Toshniwal, J. Weston, A. Szlam, and S. Sukhbaatar. "Learning to Reason and Memorize with Self-Notes". In: *CoRR* abs/2305.00833 (2023). DOI: 10.48550/arXiv.2305.00833. arXiv: 2305.00833. URL: https://doi.org/10.48550/arXiv.2305.00833.

[124] H. Zhou, A. Nova, H. Larochelle, A. C. Courville, B. Neyshabur, and H. Sedghi. "Teaching Algorithmic Reasoning via In-context Learning". In: *CoRR* abs/2211.09066 (2022). DOI: 10.48550/arXiv.2211.09066. arXiv: 2211.09066. URL: https://doi.org/10.48550/arXiv.2211.09066.

[125] S. Cognolato and A. Testolin. "Transformers discover an elementary calculation system exploiting local attention and grid-like problem representation". In: *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2022, pp. 1–8.

[126] H. Touvron et al. "Llama 2: Open Foundation and Fine-Tuned Chat Models". In: *CoRR* abs/2307.09288 (2023). DOI: 10.48550/ARXIV.2307.09288. arXiv: 2307.09288. URL: https://doi.org/10.48550/arXiv.2307.09288.

[127] X. Yue, X. Qu, G. Zhang, Y. Fu, W. Huang, H. Sun, Y. Su, and W. Chen. "MAmmoTH: Building Math Generalist Models through Hybrid Instruction Tuning". In: *CoRR* abs/2309.05653 (2023). DOI: 10.48550/ARXIV.2309.05653. arXiv: 2309.05653. URL: https://doi.org/10.48550/arXiv.2309.05653.

[128] L. Yu, W. Jiang, H. Shi, J. Yu, Z. Liu, Y. Zhang, J. T. Kwok, Z. Li, A. Weller, and W. Liu. "MetaMath: Bootstrap Your Own Mathematical Questions for Large Language Models". In: *CoRR* abs/2309.12284 (2023). DOI: 10.48550/ARXIV.2309.12284. arXiv: 2309.12284. URL: https://doi.org/10.48550/arXiv.2309.12284.

[129] R. F. Nogueira, Z. Jiang, and J. Lin. "Investigating the Limitations of the Transformers with Simple Arithmetic Tasks". In: *CoRR* abs/2102.13019 (2021). arXiv: 2102.13019. URL: https://arxiv.org/abs/2102.13019.

[130] I. R. McKenzie et al. "Inverse Scaling: When Bigger Isn't Better". In: *CoRR* abs/2306.09479 (2023). DOI: 10.48550/ARXIV.2306.09479. arXiv: 2306.09479. URL: https://doi.org/10.48550/arXiv.2306.09479.

[131] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. "GPT-4 technical report". In: *arXiv preprint arXiv:2303.08774* (2023).

[132] A. Kong, S. Zhao, H. Chen, Q. Li, Y. Qin, R. Sun, and X. Zhou. "Better Zero-Shot Reasoning with Role-Play Prompting". In: *CoRR* abs/2308.07702 (2023). DOI: 10.48550/arXiv.2308.07702. arXiv: 2308.07702. URL: https://doi.org/10.48550/arXiv.2308.07702.

[133] F. Baader and T. Nipkow. *Term rewriting and all that*. Cambridge University Press, 1998. ISBN: 978-0-521-45520-6.

[134] D. Hendrycks et al. "The Many Faces of Robustness: A Critical Analysis of Out-of-Distribution Generalization". In: *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*. IEEE, 2021, pp. 8320–8329. DOI: 10.1109/ICCV48922.2021.00823. URL: https://doi.org/10.1109/ICCV48922.2021.00823.

[135] H. Ye, C. Xie, T. Cai, R. Li, Z. Li, and L. Wang. "Towards a Theoretical Framework of Out-of-Distribution Generalization". In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. Ed. by M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan. 2021, pp. 23519–23531. URL: https://proceedings.neurips.cc/paper/2021/hash/c5c1cb0bebd56ae38817b251ad72bedb-Abstract.html.

[136] P. Velickovic, R. Ying, M. Padovano, R. Hadsell, and C. Blundell. "Neural Execution of Graph Algorithms". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=SkgKO0EtvS.

[137] R. Caruana. "Multitask learning". In: *Machine learning* 28 (1997), pp. 41–75.

[138] OpenAI. *GPT-4 Technical Report*. 2023. arXiv: 2303.08774 [cs.CL].

[139] OpenAI. *Learning to Reason with LLMs*. https://openai.com/index/learning-to-reason-with-llms/. Accessed: 2024-09-29. 2024.

[140] T. Marghetis, D. Landy, and R. L. Goldstone. "Mastering algebra retrains the visual system to perceive hierarchical structure in equations". In: *Cognitive research: principles and implications* 1 (2016), pp. 1–10.

[141] A. Kazemnejad, I. Padhi, K. N. Ramamurthy, P. Das, and S. Reddy. "The Impact of Positional Encoding on Length Generalization in Transformers". In: *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*. Ed. by A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine. 2023. URL: http://papers.nips.cc/paper%5C_files/paper/2023/hash/4e85362c02172c0c6567ce593122d31c-Abstract-Conference.html.

[142] S. Mirzadeh, K. Alizadeh, H. Shahrokhi, O. Tuzel, S. Bengio, and M. Farajtabar. "GSM-Symbolic: Understanding the Limitations of Mathematical Reasoning in Large Language Models". In: *CoRR* abs/2410.05229 (2024). DOI: 10.48550/ARXIV.2410.05229. arXiv: 2410.05229. URL: https://doi.org/10.48550/arXiv.2410.05229.