Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων
Εργαστηριο Συστηματων Τεχνητης Νοημοσυνης και Μαθησης

Investigating the Capabilities of Language Models in Puzzle Reasoning: A Survey and Experimental Analysis

Diploma Thesis

by

Panagiotis Giadikiaroglou

Επιβλέπων: Γεώργιος Στάμου
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβρης 2024

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Συστημάτων Τεχνητής Νοημοσύνης και Μάθησης

Investigating the Capabilities of Language Models in Puzzle Reasoning: A Survey and Experimental Analysis

Diploma Thesis

by

Panagiotis Giadikiaroglou

Επιβλέπων: Γεώργιος Στάμου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή επιτροπή την 24η Οκτώβριος, 2024.

....	....	....
Γεώργιος Στάμου	Αθανάσιος Βουλόδημος	Α.-Γ. Σταφυλοπάτης
Καθηγητής Ε.Μ.Π.	Επ. Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβρης 2024

....

Παναγιωτης Γιαδικιαρογλου

Διπλωματούχος Ηλεκτρολόγος Μηχανικός
και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Η επίλυση γρίφων αποτελεί σημείο αναφοράς για την αξιολόγηση των μοντέλων τεχνητής νοημοσύνης, δοκιμάζοντας την ικανότητα τους να σκέφτονται, να συμπεραίνουν και να καταστρώνουν στρατηγικές σε πολύπλοκους χώρους προβλημάτων. Οι παραδοσιακές μέθοδοι τεχνητής νοημοσύνης και μηχανικής μάθησης, όπως η συμβολική συλλογιστική και η ενισχυτική μάθηση, έχουν σημειώσει αξιοσημείωτα βήματα σε δομημένους τομείς όπως τα επιτραπέζια παιχνίδια και οι λογικοί γρίφοι. Ωστόσο, καθώς εξελίχθηκαν τα νευρωνικά δίκτυα και, πιο πρόσφατα, τα μεγάλα γλωσσικά μοντέλα (ΜΓΜ), προέκυψαν νέες δυνατότητες για την αντιμετώπιση ενός ευρύτερου φάσματος τύπων γρίφων, συμπεριλαμβανομένων εκείνων που απαιτούν λεπτή συλλογιστική κοινής λογικής, αφηρημένη αναγνώριση προτύπων και πολύπλοκους υπολογισμούς πολλών βημάτων. Τα ΜΓΜ, με τις γλωσσικές δυνατότητες που βασίζονται σε τεράστιους όγκους δεδομένων, διαθέτουν μοναδικές δυνατότητες για τη γεφύρωση δομημένων λογικών εργασιών και λιγότερο τυπικών γρίφων που βασίζονται σε κοινή γνώση. Παρά τις προόδους αυτές, το σημερινό τοπίο της επίλυσης γρίφων με ΜΓΜ αποκαλύπτει τόσο επιτεύγματα όσο και περιορισμούς, ιδίως όταν τα μοντέλα επιφορτίζονται με προβλήματα που απαιτούν ερμηνευτική συλλογιστική και ακριβείς υπολογισμούς.

Η παρούσα διπλωματική εργασία διερευνά τον εξελισσόμενο ρόλο των ΜΓΜ στην επίλυση τέτοιων σύνθετων συλλογιστικών προβλημάτων, εστιάζοντας ειδικά στις ικανότητές τους στην επίλυση γρίφων. Χωρισμένη σε δύο κύριες ενότητες, η διπλωματική παρέχει αρχικά μια ολοκληρωμένη επισκόπηση των πρόσφατων εξελίξεων στις μεθοδολογίες ΜΓΜ, καλύπτοντας ποικίλες τεχνικές προτροπής, νευροσυμβολικές προσεγγίσεις και στρατηγικές τελειοποίησης για παζλ. Χρησιμοποιώντας μια νέα προτεινόμενη ταξινόμηση, τα παζλ κατηγοριοποιούνται σε προβλήματα βασισμένα σε κανόνες και προβλήαμτα χωρίς συγκεκριμένους κανόνες, ενώ κάθε κατηγορία εξετάζεται για τις μοναδικές γνωστικές απαιτήσεις της από τα ΜΓΜ. Στη δεύτερη ενότητα παρουσιάζονται πειραματικές αξιολογήσεις που πραγματοποιήθηκαν σε τέσσερα σύνολα δεδομένων - δύο σύνολα δεδομένων που βασίζονται στα μαθηματικά (GSM8K, SVAMP) και δύο σύνολα δεδομένων που εστιάζουν σε γρίφους (Game of 24 και RiddleSense). Διάφορες τεχνικές συλλογισμού, συμπεριλαμβανομένων των προτροπών εισόδου-εξόδου (IO), των μεθόδων Chain-of-Thought (CoT), Least-to-Most (LtM) και Faithful-CoT, χρησιμοποιούνται για την αξιολόγηση των επιδόσεων των ΜΓΜ. Μοντέλα διαφορετικής κλίμακας, ιδιαίτερα μικρότερα ΜΓΜ όπως η οικογένεια μοντέλων Llama-3.1 και το Mistral, δοκιμάζονται σε περιπτώσεις όπου χρησιμοποιούνται μηδενικό πλήθος παραδειγμάτων κατά την προτροπή, ένας συγκεκριμένος αριθμός παραδειγμάτων καθώς επίσης η τεχνική της αυτοσυνέπειας για να αξιολογηθεί η αποτελεσματικότητά τους στην επίλυση σύνθετων και πολλαπλών βημάτων συλλογιστικών εργασιών. Η διπλωματική εργασία παρέχει κρίσιμες πληροφορίες σχετικά με τους περιορισμούς απόδοσης των σημερινών ΜΓΜ στην επίλυση γρίφων, σημειώνοντας ιδιαίτερα ότι προηγμένες μέθοδοι συλλογισμού όπως το Faithful-CoT και οι τεχνικές μετάφρασης γρίφων αποδίδουν ασυνεχείς βελτιώσεις στα μικρότερα μοντέλα. Τέλος, σκιαγραφεί μελλοντικές ερευνητικές κατευθύνσεις, υποστηρίζοντας τη δημιουργία διευρυμένων συνόλων δεδομένων, την ανάπτυξη νευροσυμβολικών μεθόδων και την ενδεχόμενη πρόοδο στην περιοχή δημιουργίας και παραγωγής παζλ από ΜΓΜ. Η παρούσα διπλωματική εργασία αποσκοπεί στην εμβάθυνση της κατανόησης των συλλογιστικών ικανοτήτων των ΜΓΜ και στην ανάδειξη μονοπατιών για την ενίσχυση των επιδόσεών τους σε σύνθετα γνωστικά καθήκοντα.

Λέξεις-κλειδιά — Μεγάλα γλωσσικά μοντέλα, Συλλογιστική, Επίλυση Παζλ, Προτροπή, Νευροσυμβολικές Μέθοδοι

Abstract

Puzzle-solving has long served as a benchmark for evaluating artificial intelligence, testing a model's ability to reason, infer, and strategize across complex problem spaces. Traditional AI and machine learning methods, such as symbolic reasoning and reinforcement learning, have made notable strides in structured domains like board games and logic puzzles. However, as neural networks and, more recently, large language models (LLMs) have evolved, new possibilities have emerged for tackling a broader range of puzzle types, including those requiring nuanced commonsense reasoning, abstract pattern recognition, and complex multi-step calculations. LLMs, with their vast data-driven language capabilities, hold unique potential to bridge structured logical tasks and less formal, knowledge-based puzzles. Despite these advances, the current landscape of puzzle-solving with LLMs reveals both achievements and limitations, particularly when models are tasked with problems that demand interpretative reasoning and precise calculation.

This thesis explores the evolving role of LLMs in solving such complex reasoning tasks, specifically focusing on their puzzle-solving capabilities. Divided into two main sections, the thesis first provides a comprehensive survey of recent advancements in LLM methodologies, covering diverse prompting techniques, neuro-symbolic approaches, and fine-tuning strategies for puzzles. Using a newly proposed taxonomy, puzzles are categorized into rule-based and rule-less types, with each category examined for its unique cognitive demands on LLMs. The second section presents experimental evaluations conducted on four datasets—two math-based datasets (GSM8K, SVAMP) and two puzzle-focused datasets (Game of 24 and RiddleSense). Various reasoning techniques, including Input-Output (IO) prompting, Chain-of-Thought (CoT), Least-to-Most (LtM), and Faithful-CoT methods, are employed to assess LLM performance. Models of varying scales, particularly smaller LLMs like Llama-3.1 family and Mistral, are tested across settings such as zero-shot, few-shot, and self-consistency to evaluate their efficacy in solving complex and multi-step reasoning tasks. The thesis provides critical insights into the performance limitations of current LLMs in puzzle-solving, particularly noting that advanced reasoning methods like Faithful-CoT and puzzle translation techniques yield inconsistent improvements with smaller models. Finally, it outlines future research directions, advocating for expanded dataset creation, neuro-symbolic integration, and advancements in puzzle generation. This thesis aims to deepen our understanding of LLMs' reasoning abilities and highlight pathways to enhance their performance in complex cognitive tasks.

Keywords — Large Language Models, Reasoning, Puzzle Solving, Prompting, Neurosymbolic Methods

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου, κ. Γιώργο Στάμου, για την ευκαιρία που μου έδωσε και την εμπιστοσύνη που μου έδειξε στο να εκπονήσω τη διπλωματική μου εργασία στο Εργαστήριο Συστημάτων Τεχνητής Νοημοσύνης και Μάθησης, καθώς και για την πολύτιμη καθοδήγηση που μου παρείχε, κατά τη διάρκεια αυτής της διπλωματικής, αλλά και πέραν αυτής. Ταυτόχρονα, ευχαριστώ θερμά τον κ. Θάνο Βουλόδημο για την επίσης ιδιαίτερα πολύτιμη καθοδήγησή του και την εμπιστοσύνη του. Είμαι ευγνώμων τόσο για το ακαδημαϊκό, όσο και για το ηθικό πρότυπο που αποτέλεσαν αυτοί οι άνθρωποι για εμένα κατά τη διάρκεια της συνεργασίας μας. Θα ήθελα επιπλέον να ευχαριστήσω την Μαρία Λυμπεραίου και τον Γιώργο Φιλανδριανό για τη στενή συνεργασία μας, την ανεκτίμητη βοήθεια και τη συνεχή υποστήριξή τους, δίχως των οποίων η εκπόνηση αυτής της διπλωματικής δεν θα οδηγούσε σε αυτό το αποτέλεσμα.

Τέλος, θέλω να ευχαριστήσω την οικογένειά μου, η οποία στηρίζει κάθε βήμα μου, και χωρίς αυτούς δεν θα μπορούσα να είχα καταφέρει όσα έχω πετύχει, όπως επίσης και τους φίλους μου, με τους οποίους περάσαμε αμέτρητες ώρες μελέτης, συμπαράστασης, διασκέδασης και ταξιδιών, στιγμές που θα μείνουν ανεξίτηλες στη μνήμη μου.

Παναγιώτης Γιαδικιάρογλου, Οκτώβριος 2024

Contents

List of Figures

Chapter 1

Εκτεταμένη Περίληψη στα Ελληνικά

1.1 Θεωρητικό Υπόβαθρο

Τα Μεγάλα Γλωσσικά Μοντέλα (ΜΓΜ) έχουν προωθήσει σημαντικά την επεξεργασία της φυσικής γλώσσας, επιδεικνύοντας εντυπωσιακές δυνατότητες σε ένα ευρύ φάσμα εργασιών, συμπεριλαμβανομένης της παραγωγής κειμένων, της μετάφρασης και της απάντησης ερωτήσεων [84]. Μοντέλα όπως το GPT-3 [11], το GPT-4 [89] και η πρόσφατη σειρά Llama [27, 117, 118] έχουν διευρύνει τα όρια της τεχνητής νοημοσύνης (ΤΝ), επιδεικνύοντας μια μοναδική ικανότητα κατανόησης και παραγωγής κειμένου που μοιάζει με ανθρώπινο κείμενο. Πέρα από αυτές τις επιφανειακές ικανότητες, μια κρίσιμη πτυχή αυτών των μοντέλων είναι οι δυνατότητές τους για συλλογισμό - μια γνωστική διαδικασία που περιλαμβάνει την εξαγωγή συμπερασμάτων, την επίλυση προβλημάτων και την εφαρμογή λογικής για την εξαγωγή λύσεων [70, 69, 5, 22]. Η συλλογιστική είναι ζωτικής σημασίας όχι μόνο για την κατανόηση του πλαισίου αλλά και για την αντιμετώπιση σύνθετων, αφηρημένων εργασιών που απαιτούν κάτι περισσότερο από την απομνημόνευση δεδομένων.

Η επίλυση γρίφων χρησιμεύει ως ιδανικό σημείο αναφοράς για την αξιολόγηση των ικανοτήτων συλλογιστικής των γλωσσικών μοντέλων. Τα παζλ, από το σχεδιασμό τους, προκαλούν γνωστικές ικανότητες όπως η λογική σκέψη, η αναγνώριση προτύπων και η στρατηγική σκέψη. Απαιτούν από τους λύτες να επεξεργάζονται πληροφορίες, να κάνουν συνδέσεις και να εφαρμόζουν κανόνες δημιουργικά για να καταλήξουν σε λύσεις. Αυτό καθιστά τα παζλ ένα πλούσιο πεδίο για τη δοκιμή του βάθους και της ευελιξίας των συλλογιστικών ικανοτήτων ενός μοντέλου ΤΝ. Οι γρίφοι μπορούν να κατηγοριοποιηθούν σε τύπους βασισμένους σε κανόνες και χωρίς κανόνες, καθένας από τους οποίους παρουσιάζει ξεχωριστές προκλήσεις: οι γρίφοι βασισμένοι σε κανόνες, όπως το Sudoku ή το Game-of-24, βασίζονται σε σταθερούς κανόνες και δομημένα περιβάλλοντα, ενώ οι γρίφοι χωρίς κανόνες, όπως οι γρίφοι και τα παζλ προγραμματισμού, απαιτούν ευρύτερη συμπερασματική σκέψη και εφαρμογή γνώσεων [38].

Παρά την αυξανόμενη πολυπλοκότητά τους, τα ΜΓΜ εξακολουθούν να αντιμετωπίζουν αξιοσημείωτες προκλήσεις στην επίλυση γρίφων. Οι τρέχουσες προσεγγίσεις συχνά δυσκολεύονται με εργασίες που απαιτούν συλλογισμό πολλών βημάτων, χειρισμό της αβεβαιότητας ή ερμηνεία κρυφών πληροφοριών [129, 4]. Ενώ μέθοδοι όπως η προτροπή λίγων παραδειγμάτων [10] και η συλλογιστική αλυσιδωτά βήματα σκέψης (Chain-of-Thought – CoT) [127, 53] έχουν βελτιώσει την απόδοση σε ορισμένες περιπτώσεις [105], παραμένουν σημαντικά κενά, ιδιαίτερα σε γρίφους που απαιτούν βαθιά λογική συμπερασματολογία ή το συνδυασμό πολλαπλών βημάτων συλλογιστικής. Αυτοί οι περιορισμοί αναδεικνύουν την ανάγκη για πιο προηγμένες τεχνικές που μπορούν να ενισχύσουν τις διαδικασίες συλλογισμού των ΜΓΜ.

Η διερεύνηση των συλλογιστικών ικανοτήτων των ΜΓΜ μέσω της επίλυσης παζλ παρέχει πολύτιμες γνώσεις που μπορούν να οδηγήσουν στην ανάπτυξη πιο προηγμένων συστημάτων τεχνητής νοημοσύνης, εξοπλίζοντάς τα με την ικανότητα να χειρίζονται σύνθετα συλλογιστικά προβλήματα. Η κατανόηση του τρόπου με τον οποίο αυτά τα μοντέλα χειρίζονται εργασίες συλλογισμού μπορεί να βοηθήσει στην ανάπτυξη νέων στρατηγικών και να καθοδηγήσει τη δημιουργία μοντέλων καλύτερα εξοπλισμένων για την αντιμετώπιση σύνθετων προκλήσεων του πραγματικού κόσμου. Η παρούσα διπλωματική εργασία έχει ως στόχο να γεφυρώσει το χάσμα μεταξύ των σημερινών δυνατοτήτων των ΜΓΜ και των απαιτήσεων της προηγμένης συλλογιστικής, συμβάλλοντας με πολύτιμες γνώσεις στην ανάπτυξη πιο ισχυρών μεθόδων συλλογιστικής ΤΝ.

Η μελέτη μας έχει δύο κύριες συνεισφορές:

1. Μια ολοκληρωμένη επισκόπηση του υπάρχοντος τοπίου της συλλογιστικής ΜΓΜ στην επίλυση παζλ, αναδεικνύοντας τα δυνατά και αδύνατα σημεία των διαφόρων προσεγγίσεων.

2. Πειραματικές αξιολογήσεις της απόδοσης των ΜΓΜ σε επιλεγμένα παζλ, συμπεριλαμβανομένων μαθηματικών συνόλων δεδομένων, του RiddleSense [67] και του Game-of-24 Puzzle, χρησιμοποιώντας διαφορετικές στρατηγικές συλλογισμού, όπως η προτροπή λίγων παραδειγμάτων, το CoT, η αυτοσυνέπεια και οι νευροσυμβολικές προσεγγίσεις.

1.1.1 Μεγάλα Γλωσσικά Μοντέλα

Τα Μεγάλα Γλωσσικά Μοντέλα (ΜΓΜ) είναι προηγμένα συστήματα τεχνητής νοημοσύνης που χρησιμοποιούν βαθιά μάθηση για να επεξεργάζονται και να παράγουν φυσική γλώσσα, επιτυγχάνοντας επίπεδα κατανόησης και αλληλεπίδρασης που σηματοδοτούν επανάσταση στην τεχνητή νοημοσύνη. Αυτά τα μοντέλα, τα οποία εκπαιδεύονται σε εκτεταμένα σύνολα δεδομένων από πηγές όπως βιβλία και περιεχόμενο του διαδικτύου, είναι ικανά να συλλαμβάνουν πολύπλοκα γλωσσικά πρότυπα και δομές, καθιστώντας τα ιδιαίτερα ευέλικτα σε ένα ευρύ φάσμα εργασιών που σχετίζονται με τη γλώσσα. Μετά από μια ευρεία φάση προ-εκπαίδευσης, τα ΜΓΜ συχνά προσαρμόζονται λεπτομερώς για συγκεκριμένους τομείς, επιτρέποντάς τους να προσαρμόζουν τη γενική γλωσσική τους κατανόηση σε εξειδικευμένες εφαρμογές με ελάχιστα πρόσθετα δεδομένα εκπαίδευσης.

Η εξέλιξη αυτή αποτελεί ένα σημαντικό άλμα σε σχέση με τα προηγούμενα νευρωνικά μοντέλα, επιτρέποντας στα ΜΓΜ να αναλαμβάνουν καθήκοντα που υπερβαίνουν την απλή παραγωγή κειμένου και περιλαμβάνουν εξελιγμένη συλλογιστική, επίλυση προβλημάτων και δημιουργικές εφαρμογές. Αντιμετωπίζουν ένα ευρύ φάσμα προκλήσεων, από την ταξινόμηση και την περίληψη κειμένων έως τη γλωσσική μετάφραση και τη συνομιλία με την τεχνητή νοημοσύνη. Η προσαρμοστικότητα των ΜΓΜ τα καθιστά πολύτιμα σε όλους τους κλάδους, παρέχοντας λύσεις ΤΝ που ξεπερνούν τις παλαιότερες τεχνολογίες σε ταχύτητα, ακρίβεια και ευελιξία.

Ο πυρήνας των ΜΓΜ βρίσκεται στην αρχιτεκτονική του Transformer (μετασχηματιστή), η οποία παρουσιάστηκε από τους Vaswani et al. (2017) [119]. Οι Transformers βασίζονται σε μηχανισμούς αυτοπροσοχής, οι οποίοι επιτρέπουν στα μοντέλα να κατανοούν τις σχέσεις μεταξύ των λέξεων σε μια ακολουθία κειμένου, ανεξάρτητα από τη σειρά. Αυτή η αρχιτεκτονική ενισχύει την ικανότητα του μοντέλου να κατανοεί και να παράγει κείμενο με νόημα. Η εκπαίδευση των ΜΓΜ ξεκινά γενικά με μάθηση χωρίς επίβλεψη, όπου το μοντέλο προβλέπει την επόμενη λέξη σε μια ακολουθία, δημιουργώντας μια θεμελιώδη κατανόηση της γλώσσας. Αυτό συνήθως ακολουθείται από επιβλεπόμενη τελειοποίηση σε συγκεκριμένες εργασίες, βελτιώνοντας περαιτέρω τα παραγόμενα αποτελέσματα του μοντέλου.

Σε αντίθεση με προηγούμενες αρχιτεκτονικές, όπως τα Αναδρομικά Νευρωνικά Δίκτυα (RNN) [108] και τα δίκτυα μακράς βραχυπρόθεσμης μνήμης (LSTM) [43, 108], οι Transformers δεν απαιτούν διαδοχική επεξεργασία δεδομένων, καθιστώντας τους κλιμακούμενους και ιδανικούς για μεγάλα σύνολα δεδομένων. Αυτή η μετατόπιση υπήρξε το κλειδί για την επιτυχία των σύγχρονων ΜΓΜ, τα οποία αξιοποιούν τον υψηλό αριθμό παραμέτρων τους και τα εκτεταμένα δεδομένα εκπαίδευσης για να γενικεύουν αποτελεσματικά σε πολύπλοκες γλωσσικές εργασίες.

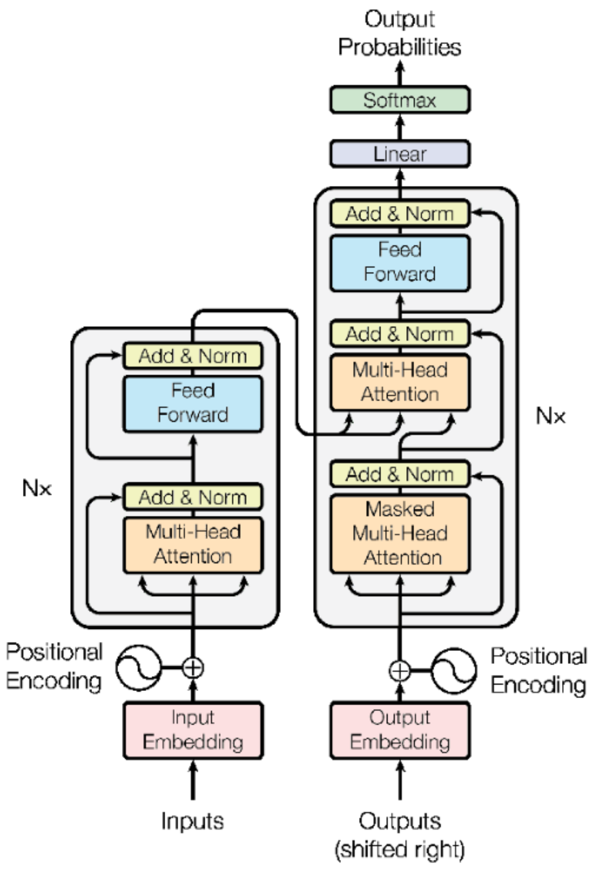


Figure 1.1.1: Αρχιτεκτονική Transformer (Μετασχηματιστή) [119]

1.1.2 Προτροπή

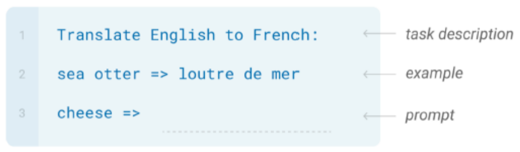
Η προτροπή χρησιμεύει ως είσοδος σε ένα παραγωγικό σύστημα τεχνητής νοημοσύνης, περιγράφοντας τη συγκεκριμένη εργασία ή στόχο που πρέπει να εκτελέσει το μοντέλο. Ουσιαστικά καθοδηγεί το μοντέλο παρέχοντας σχετικό περιεχόμενο, βοηθώντας το σύστημα να κατανοήσει το αίτημα του χρήστη. Η διαδικασία της προτροπής επιτρέπει στους χρήστες να καθοδηγούν τα ΜΓΜ με συγκεκριμένες μορφές ή οδηγίες, συχνά χωρίς την ανάγκη για λεπτομερή ρύθμιση ή τροποποίηση των παραμέτρων του μοντέλου. Η προσέγγιση αυτή προσφέρει αποτελεσματικότητα και ευελιξία, καθώς μια καλά σχεδιασμένη προτροπή μπορεί να καθοδηγήσει το μοντέλο προς τις επιθυμητές εξόδους χρησιμοποιώντας προϋπάρχουσα γνώση.

Η προτροπή παρέχει διάφορα πλεονεκτήματα, ιδίως αποτελεσματικότητα και ευελιξία. Σε αντίθεση με τις παραδοσιακές μεθόδους, οι οποίες μπορεί να απαιτούν πρόσθετα δεδομένα εκπαίδευσης ή λεπτομερή ρύθμιση των παραμέτρων, η προτροπή αποδίδει άμεσα αποτελέσματα με ελάχιστη υπολογιστική προσπάθεια. Αυτή η αποτελεσματικότητα την καθιστά ιδιαίτερα πολύτιμη όταν οι χρήστες χρειάζονται γρήγορη προσαρμογή των μοντέλων σε νέες εργασίες. Επιπλέον, η προτροπή δεν επηρεάζει τις εργασίες: με τις κατάλληλες εισόδους, τα ΜΓΜ μπορούν να εκτελέσουν ένα ευρύ φάσμα εργασιών, όπως αναφέρθηκε παραπάνω, χωρίς να απαιτούνται ξεχωριστά μοντέλα για συγκεκριμένες εργασίες.

Ωστόσο, η προτροπή έχει περιορισμούς. Οι αποτελεσματικές προτροπές βασίζονται στην υπάρχουσα γνώση του μοντέλου, η οποία μπορεί να είναι ελλιπής ή προκατειλημμένη. Η σύνταξη μιας καλά δομημένης προτροπής μπορεί επίσης να αποτελέσει πρόκληση, καθώς μικρές παραλλαγές στη διατύπωση μπορούν να επηρεάσουν σημαντικά την απόδοση του μοντέλου. Αυτό έχει οδηγήσει στην ανάπτυξη της μηχανικής προτροπών, ενός τομέα που επικεντρώνεται στη βελτίωση των προτροπών για βέλτιστα αποτελέσματα. Ενώ η προτροπή μπορεί να προσαρμόσει ένα μοντέλο σε διάφορες εργασίες, τα λεπτομερώς ρυθμισμένα μοντέλα συχνά παρέχουν πιο ακριβή αποτελέσματα για πολύ συγκεκριμένες εφαρμογές.

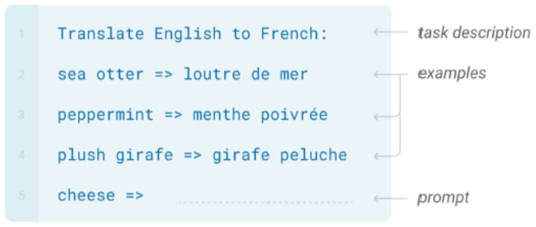
Η επιτυχία της προτροπής εξαρτάται από παράγοντες όπως η σαφήνεια της προτροπής, η ευθυγράμμιση με την εκπαίδευση του μοντέλου και η συμπερίληψη σχετικών παραδειγμάτων. Παρά τους περιορισμούς της, η προτροπή παραμένει ένας επεκτάσιμος τρόπος για την αξιοποίηση των δυνατοτήτων των ΜΓΜ σε ποικίλες εφαρμογές, ιδίως σε τομείς όπως η επεξεργασία φυσικής γλώσσας και η πολυτροπική μάθηση. Διάφοροι τύποι και τεχνικές προτροπής, όπως η προτροπή με μηδενικό πλήθος παραδειγμάτων, η προτροπή με λίγα παραδείγματα και η προτροπή με αλυσιδωτά βήματα σκέψης (CoT), βελτιστοποιούν την απόδοση των ΜΓΜ και προσαρμόζονται σε διαφορετικές εργασίες.

Η προτροπή με αλυσιδωτά βήματα σκέψης (CoT) είναι μια τεχνική που ενθαρρύνει τα Μεγάλα Γλωσσικά Μοντέλα να παράγουν απαντήσεις αναλύοντας τη διαδικασία συλλογισμού σε τμήματα βήμα προς βήμα, τα οποία οδηγούν σε μια τελική απάντηση. Η προσέγγιση αυτή βελτιώνει την απόδοση του μοντέλου σε σύνθετες εργασίες όπως η επίλυση γρίφων ή μαθηματικών προβλημάτων, όπου η λογική εξέλιξη είναι απαραίτητη. Σε αντίθεση με την προτροπή μηδενικών παραδειγμάτων, όπου το μοντέλο δεν λαμβάνει παραδείγματα, ή την προτροπή λίγων παραδειγμάτων, η οποία παρέχει μερικά παραδείγματα για να καθοδηγήσει το μοντέλο, το Chain-of-Thought δομεί ρητά τη διαδικασία σκέψης του μοντέλου, ενισχύοντας τόσο την ακρίβεια όσο και την ερμηνευσιμότητα, καθιστώντας κάθε βήμα συλλογισμού ξεκάθαρο.



```
Translate English to French:
sea otter => loutre de mer
cheese =>
```

(a) Προτροπή με ένα παράδειγμα [10]



```
Translate English to French:
sea otter => loutre de mer
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>
```

(b) Προτροπή με λίγα παραδείγματα [10]

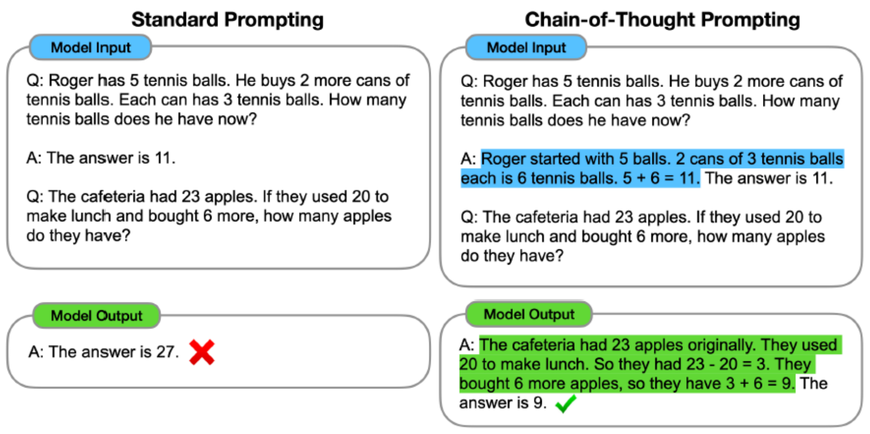

Standard Prompting
Model Input
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: The answer is 11.
Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?
Model Output
A: The answer is 27.
Chain-of-Thought Prompting
Model Input
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.
Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?
Model Output
A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9.

Figure 1.1.3: Προτροπή με αλυσιδωτά βήματα σκέψης (CoT) [127]

1.2 Επίλυση Παζλ με χρήση ΜΓΜ (Βιβλιογραφική Έρευνα)

Αυτή η ενότητα παρουσιάζει μια εκτεταμένη διευρεύνηση των δυνατοτήτων επίλυσης γρίφων στα Μεγάλα Γλωσσικά Μοντέλα (ΜΓΜ), επεκτείνοντας τη ανασκόπηση της βιβλιογραφίας που παρουσιάστηκε στην εργασία μας *Puzzle Solving using Large Language Models: A Survey* (Giadikiaroglou et al., 2024) [38]. Θα παράσχουμε μια αναλυτική συζήτηση για τους τύπους γρίφων, τις μεθόδους, τα σύνολα δεδομένων που χρησιμοποιούνται για την αξιολόγηση των συλλογιστικών ικανοτήτων των ΜΓΜ αυτόν τον τομέα.

1.2.1 Κατηγοριοποίηση των Παζλ

Για την αξιολόγηση των ικανοτήτων συλλογισμού των ΜΓΜ, είναι σημαντικό να κατηγοριοποιήσουμε τα παζλ. Διακρίνουμε τα παζλ ανάλογα με την εξάρτησή τους από τυπικούς κανόνες ή την ευρύτερη γνώση του κόσμου που συνοδεύεται από γενικές ικανότητες εξαγωγής συμπερασμάτων, όπως απεικονίζεται στο Σχήμα 1.2.1. Αυτή η κατηγοριοποίηση όχι μόνο αναδεικνύει τη γνωστική ποικιλομορφία που παρουσιάζουν τα παζλ, αλλά και ευθυγραμμίζεται με τις ξεχωριστές προκλήσεις συλλογισμού: τα παζλ που βασίζονται σε κανόνες απαιτούν λογική εξαγωγή συμπερασμάτων και στρατηγική πρόβλεψη εντός κλειστών περιβαλλόντων με καθορισμένες παραμέτρους, ενώ τα παζλ χωρίς κανόνες απαιτούν γενικές ικανότητες συλλογισμού, ερμηνεία καταστάσεων και εξήγηση γεγονότων με την εξαγωγή συμπερασμάτων που βασίζονται σε πρακτικές γνώσεις για τον καθημερινό κόσμο.

Παζλ Βασιζόμενα σε Κανόνες

Τα βασιζόμενα σε κανόνες παζλ παρέχουν στο μοντέλο ρητές συνθήκες νίκης, σύνολα νόμιμων κινήσεων ή κανόνες μετάβασης κατάστασεων. Υποδιαιρούμε περαιτέρω αυτή την κατηγορία με βάση το αν οι μεταβάσεις καταστάσεων είναι ντετερμινιστικές ή ενσωματώνουν τυχαιότητα.

Ντετερμινιστικά Παίγνια: παράγουν πάντα την ίδια διάδοχη κατάσταση δεδομένης της τρέχουσας κατάστασης του παιγνίου και της δράσης που λαμβάνεται σύμφωνα με τους κανόνες. Για παράδειγμα, στο Σκάκι, η πραγματοποίηση μιας κίνησης παράγει πάντα μια μονοσήμαντη νέα διάταξη της σκακιέρας. Άλλα παραδείγματα περιλαμβάνουν το Sudoku, την πλοήγηση σε λαβύρινθο ή την επίλυση του κύβου του Rubik. Το μοντέλο θα πρέπει να μαθαίνει στρατηγικές που λειτουργούν εντός του χώρου δυνατοτήτων που ορίζεται από τους νόμιμους μηχανισμούς του παιγνίου.

Στοχαστικά παίγνια: ενσωματώνουν τυχαιότητα ή κρυφή πληροφορία, δηλαδή η ίδια ενέργεια του παίκτη μπορεί να οδηγήσει σε διαφορετικές κατανομές πιθανοτήτων για τις επόμενες καταστάσεις. Παραδείγματα περιλαμβάνουν τον Ναρκαλιευτή (Minesweeper) (κρυφές τοποθεσίες βομβών) ή παιχνίδια με χαρτιά, π.χ. το πόκερ, όπου οι αντίπαλοι κρατούν κρυφά τα φύλλα τους. Η γνώση αυτών των παιγνίων απαιτεί συλλογισμό πάνω σε αβέβαιες καταστάσεις, σχεδιασμό πολλαπλών κινήσεων εκ των προτέρων και διαχείριση ρίσκου.

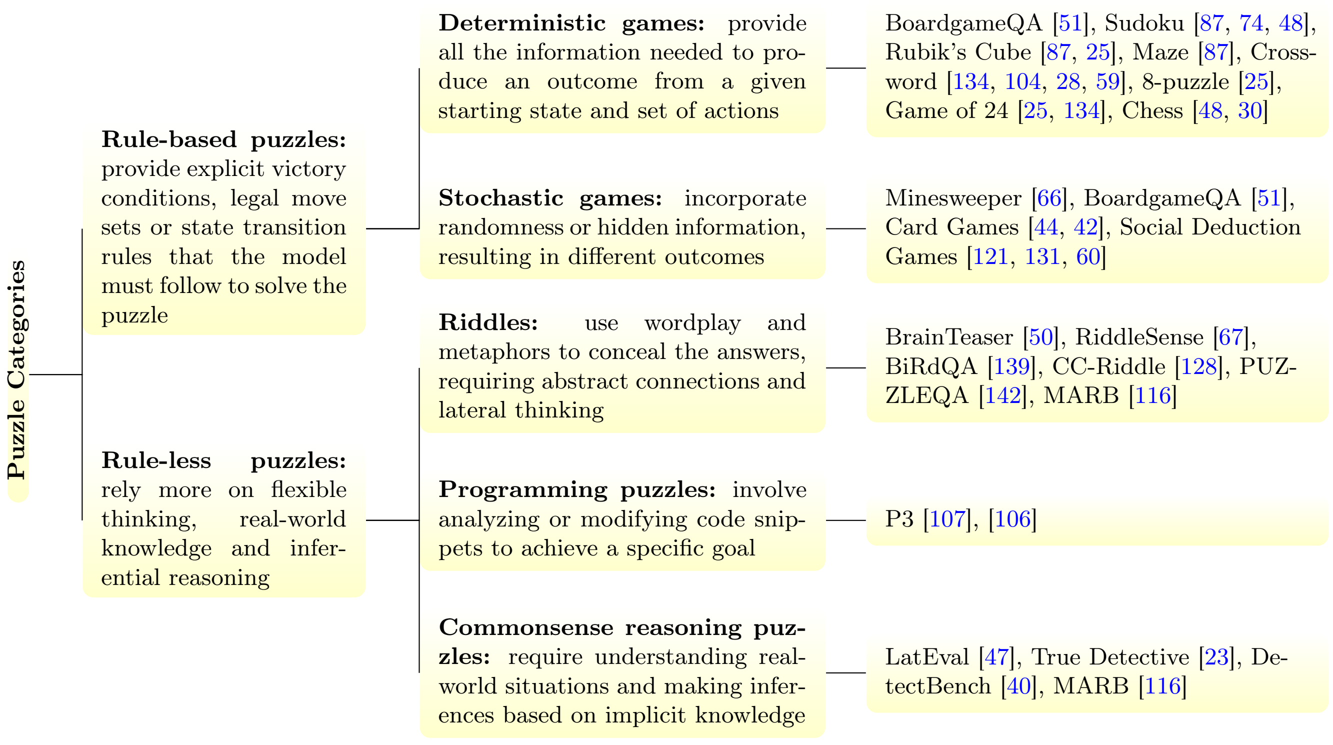

Puzzle Categories
Rule-based puzzles: provide explicit victory conditions, legal move sets or state transition rules that the model must follow to solve the puzzle
Rule-less puzzles: rely more on flexible thinking, real-world knowledge and inferential reasoning
Deterministic games: provide all the information needed to produce an outcome from a given starting state and set of actions
Stochastic games: incorporate randomness or hidden information, resulting in different outcomes
Riddles: use wordplay and metaphors to conceal the answers, requiring abstract connections and lateral thinking
Programming puzzles: involve analyzing or modifying code snippets to achieve a specific goal
Commonsense reasoning puzzles: require understanding real-world situations and making inferences based on implicit knowledge
BoardgameQA [51], Sudoku [87, 74, 48], Rubik's Cube [87, 25], Maze [87], Crossword [134, 104, 28, 59], 8-puzzle [25], Game of 24 [25, 134], Chess [48, 30]
Minesweeper [66], BoardgameQA [51], Card Games [44, 42], Social Deduction Games [121, 131, 60]
BrainTeaser [50], RiddleSense [67], BiRdQA [139], CC-Riddle [128], PUZZLEQA [142], MARB [116]
P3 [107], [106]
LatEval [47], True Detective [23], DetectBench [40], MARB [116]

Figure 1.2.1: Ταξινόμηση των Παζλ με τα αντίστοιχα Σύνολα Δεδομένων.

Παζλ Μη-Βασιζόμενα σε Κανόνες

Σε αντίθεση με τα παζλ με κανόνες, τα προβλήματα χωρίς κανόνες βασίζονται περισσότερο στην ευέλικτη σκέψη και τη γνώση του πραγματικού κόσμου για την ερμηνεία ασαφών καταστάσεων και την εξαγωγή συμπερασμάτων για μη παρατηρημένες λεπτομέρειες. Αντί να δοκιμάζουν τη συστηματική αναζήτηση ή τον στρατηγικό σχεδιασμό, αυτά τα παζλ μετρούν τις γνωστικές δεξιότητες για την ερμηνεία του περιεχομένου, τον συνδυασμό εννοιών και τη συλλογιστική με βάση την κοινή λογική. Στην κατηγορία αυτή εμπίπτουν τα ακόλουθα.

Οι **Γρίφοι** χρησιμοποιούν λογοπαίγνια για να αποκρύψουν τις απαντήσεις. Για παράδειγμα, το ερώτημα «Τι γίνεται πιο υγρό όσο πιο πολύ στεγνώνει;» έχει τη λύση «μια πετσέτα» μέσω της μεταφορικής έννοιας. Η επίλυση αινιγμάτων απαιτεί την πραγματοποίηση αφηρημένων συνδέσεων μεταξύ εννοιών που κρύβονται σε λυρική γλώσσα.

Τα **Προγραμματιστικά Παζλ** παρέχουν αποσπάσματα κώδικα και απαιτούν ανάλυση ή τροποποίηση της λογικής του προγράμματος. Ο Schuster (2021) [107] ορίζει ένα παζλ προγραμματισμού ως ένα σύντομο πρόγραμμα Python f, και ο στόχος είναι να βρεθεί μια είσοδος που κάνει το f να επιστρέφει True. Τέτοια παζλ αξιολογούν δεξιότητες όπως η ανίχνευση της εκτέλεσης, η διόρθωση σφαλμάτων ή η πρόβλεψη εξόδων με βάση τη σημασιολογία κωδικοποίησης. Για παράδειγμα, ο παρακάτω γρίφος ελέγχει την κατανόηση της σημασιολογίας προγραμματισμού για την πρόβλεψη της συμπεριφοράς ενός συστήματος:

```
def mystery(x):
    return x // 2
print(mystery(10))
```

Τα **Παζλ κοινής λογικής** απεικονίζουν τυπικές καταστάσεις παραλείποντας βασικές λεπτομέρειες. Οι λύτες πρέπει να εξηγήσουν τα γεγονότα συμπεραίνοντας αληθοφανείς υποθέσεις σχετικά με τα κίνητρα, τις αιτίες και τα αποτελέσματα. Για παράδειγμα, η ερώτηση «Ένας άνδρας που βρισκόταν έξω στη βροχή χωρίς ομπρέλα ή καπέλο δεν έβρεξε ούτε μια τρίχα στο κεφάλι του. Γιατί;» κρύβει την πληροφορία ότι ο άνδρας είναι καραφλός.

1.2.2 Μεθοδολογίες Επίλυσης Παζλ

Η χρήση των ΜΓΜ για την επίλυση παζλ περιλαμβάνει ένα ευρύ φάσμα μεθόδων και στρατηγικών που ενισχύει την πολύπλοκη συλλογιστική και τις επιδόσεις τους. Διαχωρίζουμε τις υπάρχουσες μεθόδους σε τεχνικές προτροπής, νευροσυμβολικές προσεγγίσεις για τη μετάφραση παζλ και στη λεπτομερή ρύθμιση (Fine-tuning) για συγκεκριμένους τομείς. Μια λεπτομερής επισκόπηση των μεθόδων που χρησιμοποιούνται σε διάφορες κατηγορίες παζλ παρουσιάζεται στον πίνακα 1.1.

Methods	**Rule-based Puzzles**		**Rule-less Puzzles**		
	Deterministic	**Stochastic**	**Riddles**	**Programming**	**Commonsense**
Prompting	-	-	-	-	-
Few-shot	✓	✓	✓	✓	✓
Chain-of-Thought	✓	✓	✓	✓	✓
Self-refine	✓				
Auto-CoT					✓
Complexity CoT					✓
Plan & Solve					✓
Detective Thinking					✓
Self-Consistency	✓				✓
Tree-of-Thoughts	✓				
Tree-of-uncertain-Thoughts	✓				
Inferential Exclusion Prompting			✓		✓
Graph-of-Thoughts	✓				
Everything-of-thoughts	✓				
Hints			✓		✓
Introduction/Summarization	✓	✓	✓	✓	✓
Puzzle Translation	-	-	-	-	-
Logic	✓				
Code					
Fine-Tuning	✓	✓	✓	✓	✓

Table 1.1: Μέθοδοι για την κάθε κατηγορία της ταξινόμησης μας με βάση τα σύνολα δεδομένων που συλλέχθηκαν.

Τεχνικές Προτροπής

Οι στρατηγικές προτροπής που παρέχουν ενδιάμεσα βήματα συλλογισμού είναι κρίσιμης σημασίας για την ενίσχυση των δυνατοτήτων επίλυσης παζλ των γλωσσικών μοντέλων. Το παράδειγμα **few-shot in-context learning** προσφέρει ένα ή περισσότερα παραδείγματα μέσα σε προτροπές, βελτιώνοντας σημαντικά τις επιδόσεις τόσο για γρίφους που βασίζονται σε κανόνες όσο και για γρίφους χωρίς κανόνες, παρουσιάζοντας τη διαδικασία συλλογισμού χωρίς πρόσθετη εκπαίδευση [11, 26, 144]. Πρόσφατες εργασίες επικεντρώνονται στο πώς διαφορετικές «δομές σκέψης» μπορούν να καθοδηγήσουν τα ΜΓΜ στην τελική λύση [8].

Αλυσιδωτές Δομές, οι οποίες περιλαμβάνουν το **Chain-of-Thought (CoT)** [125, 53] έχουν εφαρμοστεί σε όλα τα είδη παζλ, αποδεικνύοντας την υπεροχή τους σε σχέση με τις απλές προτροπές εισόδου-εξόδου. **Self-Refine** [79] χρησιμοποιείται για το παζλ Game of 24 (βασισμένο σε κανόνες/ντετερμινιστικό), ξεπερνώντας το CoT με 13% υψηλότερο ποσοστό επιτυχίας [134]. Οι Gu et al. (2023) [40] χρησιμοποιούν διάφορες μεθόδους σε ένα σύνολο δεδομένων τύπου ντετέκτιβ χωρίς κανόνες, συμπεριλαμβανομένου του **Automatic CoT**, το οποίο παράγει αυτόματα διάφορες αλυσίδες συλλογισμού για διάφορες ερωτήσεις [140], το **Complexity CoT** που αξιοποιεί την πολυπλοκότητα των ζητούμενων αλυσίδων, όπου τα πιο περίπλοκα βήματα συλλογισμού συχνά οδηγούν σε βελτιωμένη απόδοση σε σύνθετες εργασίες συμπερασμού, επιλέγοντας αποτελέσματα που καταδεικνύουν βαθύτερες ικανότητες συλλογισμού [33], και τη μέθοδο **Plan-and-Solve (PS)**, η οποία χρησιμοποιεί δύο προτροπές για κάθε πρόβλημα - μία για να δημιουργήσει τη διαδικασία συλλογισμού και την αντίστοιχη απάντηση και μία άλλη για να εξάγει την τελική απάντηση από την αρχική παραγωγή [120]. Παρά τις ποικίλες προσεγγίσεις, καμία από αυτές τις μεθόδους δεν ξεπέρασε ξεκάθαρα το CoT σε όλες τις δοκιμασμένα ΜΓΜ. Τα καλύτερα αποτελέσματα επιτυγχάνονται από την μέθοδο **Detective Thinking Prompt**, μια τεχνική που μοιάζει με το CoT και παρουσιάστηκε στην ίδια μελέτη, η οποία δεν ξεπερνά την ακρίβεια του 61.6% του καλύτερου μοντέλου, GPT-4. Η μέθοδος ενθαρρύνει το μοντέλο να εξετάζει και να αναλύει πολλαπλές ενδείξεις σε ένα δεδομένο σενάριο, δημιουργώντας διαδοχικά ένα συμπέρασμα. Αυτός ο τύπος προτροπής μπορεί να βοηθήσει το μοντέλο να χειριστεί πολύπλοκα σενάρια όπου η σωστή σύνθεση διαφορετικών πληροφοριών είναι κρίσιμης σημασίας για τη δημιουργία ακριβών και λογικών αποτελεσμάτων. Ο Schuster (2021) [107] χρησιμοποίησε αποκλειστικά τις λύσεις σε προγραμματιστικά παζλ που το μοντέλο είχε ήδη λύσει ως παραδείγματα, ξεπερνώντας τις εναλλακτικές προσεγγίσεις.

Δενδρικές Δομές, οι οποίες καλύπτουν μια ποικιλία μεθόδων. **Self-Consistency (SC)** [123] έχει δοκιμαστεί σε ντετερμινιστικά παζλ βασισμένα σε κανόνες, όπως το 8-puzzle, το Game of 24 και το Pocket Cube, καθώς και σε παζλ κοινής λογικής χωρίς κανόνες, παρουσιάζοντας μια μικρή βελτίωση στην πρώτη κατηγορία έναντι του CoT [25, 134, 86] και κανένα ξεκάθαρο όφελος στη δεύτερη κατηγορία [40]. Το **Tree-of-Thought(s) (ToT)** [134, 74] έχει εφαρμοστεί αποκλειστικά σε ντετερμινιστικά παζλ βασισμένα σε κανόνες μέχρι στιγμής, με σημαντικά βελτιωμένα ποσοστά επιτυχίας σε σχέση με το CoT, με αυξήσεις που κυμαίνονται από 26% [86] έως 70% [134] ανάλογα με τον γρίφο και το βάθος του δέντρου, παρά τις αυξημένες κλήσεις σε ΜΓΜ [25]. Το **Tree-of-Uncertain-Thought (TouT)** [86] πέτυχε ακόμη καλύτερα αποτελέσματα από το ToT στις ίδιες προκλήσεις, με 9% υψηλότερο ποσοστό επιτυχίας στο Game of 24 και 3% στα σταυρόλεξα. Τέλος, η μέθοδος **Inference-Exclusion-Prompting (IEP)** [116] χρησιμοποιεί έναν συνδυασμό εμπρόσθιας και οπίσθιας συλλογιστικής για να προσεγγίσει την ανθρώπινη λογική και έδωσε μερικά από τα καλύτερα αποτελέσματα σε γρίφους και παζλ κοινής λογικής όταν συνδυάστηκε με το CoT, σημειώνοντας 82% σε παζλ κοινής λογικής - από 81% με CoT χωρίς παραδείγματα - και 79% σε γρίφους, σε σύγκριση με 82% με CoT χωρίς παραδείγματα.

Δομές Γράφων, οι οποίες συνεπάγονται τα ακόλουθα: **Graph-of-Thought(s) (GoT)** [7, 62] και **Everything-of-Thought (XoT)** [25] έχουν χρησιμοποιηθεί για την επίλυση ντετερμινιστικών παζλ βασισμένων σε κανόνες. Ενώ η μέθοδος GoT έχει δείξει χειρότερα αποτελέσματα σε σύγκριση με την ToT, με μείωση που κυμαίνεται από 2% έως 6% [25], η μέθοδος XoT έχει αναγνωριστεί ως η πιο αποτελεσματική μέθοδος για αυτούς τους γρίφους. Το XoT ενσωματώνει την αναζήτηση δέντρων Monte Carlo (Monte Carlo Tree Search – MCTS) με ΜΓΜ για βελτιωμένη παραγωγή σκέψεων, επιτυγχάνοντας βελτιώσεις στα αποτελέσματα από 53% έως 69% σε σύγκριση με την ToT. Επιπλέον, η XoT παρουσιάζει τις λιγότερες κλήσεις ΜΓΜ μεταξύ των μεθόδων που δοκιμάστηκαν, συμπεριλαμβανομένων των CoT, SC, ToT και GoT.

Μετάφραση Παζλ

Οι μέθοδοι μετάφρασεις παζλ, αποτελούν **νευροσυμβολικές τεχνικές** που χρησιμοποιούνται από τα ΜΓΜ για τη μετάφραση γρίφων κειμένου από τη φυσική γλώσσα σε μορφές που επιδέχονται πιο εύκολα λύσεις από εξωτερικά εργαλεία. Αξίζει να σημειωθεί ότι αυτές οι μέθοδοι δεν ελέγχουν την ικανότητα των ΜΓΜ να λύνουν γρίφους αλλά αξιολογούν την ικανότητά τους να κωδικοποιούν γρίφους σε κατάλληλες αναπαραστάσεις.

Η αρχική προσέγγιση περιλαμβάνει τη χρήση ΜΓΜ για τη δημιουργία **λογικών κανόνων (1ης ή 2ης τάξης)** από τη φυσική γλώσσα του παζλ και στη συνέχεια την επίλυσή του με τη χρήση ενός εξωτερικού εργαλείου επίλυσης λογικών κανόνων. Οι Ishay et al. (2023) [48] χρησιμοποιούν τα GPT-3 και GPT-4 για να μετατρέψουν λογικά παζλ, όπως γρίφους σκακιού, Jobs Puzzles και Sudoku (ντετερμινιστικά παζλ βασισμένα σε κανόνες) σε μορφή Answer Set Programs (ASP) με τη δημιουργία κατηγορημάτων και κανόνων. Αποδεικνύουν ότι αυτή η μέθοδος πέτυχε σημαντικά αποτελέσματα, με το GPT-4 να σημειώνει ακρίβεια 92% σε ένα σύνολο δεδομένων λογικών γρίφων [85], σε σύγκριση με 7% σε προτροπή με λίγα παραδείγματα και 21% σε προτροπή χωρίς παραδείγματα με το ίδιο μοντέλο. Σημειώνουν ότι σε προτροπή με λίγα παραδείγματα, τα ΜΓΜ μπορούν να δημιουργήσουν πολύπλοκα προγράμματα που οι άνθρωποι μπορούν εύκολα να βελτιώσουν και να διορθώσουν σε περίπτωση σφαλμάτων στον κώδικα. Επιπλέον, παρόμοια πλαίσια όπως το Logic-LM [90], το LINC [88] και η μέθοδος των Yang et al. (2023) [132] δείχνουν πολλά υποσχόμενα αποτελέσματα σε διεργασίες που απαιτούν λογική συλλογιστική.

Ενώ νευροσυμβολικές προσεγγίσεις έχουν εφαρμοστεί στη μετάφραση παζλ σε λογικούς κανόνες, στη διάρκεια της συγγραφής αυτής της διπλωματικής εργασίας δεν έχουμε βρει μελέτες για τη μετατροπή γρίφων από φυσική γλώσσα σε **κώδικα**. Ωστόσο, τεχνικές όπως η προτροπή Program of Thoughts (PoT) [16], Program-Aided

Language (PAL) [35] και Faithful-CoT [78] χρησιμοποιούν μοντέλα για τη μετατροπή συλλογισμών σε προγράμματα Python για σύνολα δεδομένων λογικής και μαθηματικής συλλογιστικής. Ως εκ τούτου, ενθαρρύνουμε την ερευνητική κοινότητα να διερευνήσει αυτές τις μεθόδους και για προβλήματα επίλυσης παζλ.

Fine-Tuning

Η λεπτομερής ρύθμιση των ΜΓΜ (Fine-Tuning) αναδεικνύεται ως μια ισχυρή στρατηγική για την ενίσχυση των συλλογιστικών τους ικανοτήτων, που κυμαίνονται από τη γενική λογική συλλογιστική έως τις ειδικές ικανότητες επίλυσης παζλ.

Παζλ Βασιζόμενα σε Κανόνες Στον τομέα των ντετερμινιστικών παζλ που βασίζονται σε κανόνες, [87] παρατηρούμε υποβέλτιστα αποτελέσματα κατά το fine-tuning του GPT-2 στο Sudoku, τον κύβο του Rubik και τους λαβύρινθους, ενδεχομένως λόγω της σύντομης περιόδου fine-tuning και των περιορισμένων παραδειγμάτων εκπαίδευσης. Όσον αφορά τα σταυρόλεξα, διάφορες μελέτες [104, 28] παρουσιάζουν μικτά αποτελέσματα, με ορισμένα fine-tuned ΜΓΜ να υπερτερούν έναντι των μη νευρωνικών λύσεων, ενώ κάποια άλλα όχι, υπογραμμίζοντας την εγγενή πρόκληση των κρυπτόλεξων για τα ΜΓΜ. Οι Kazemi et al. (2023) [51] αποδεικνύουν ότι η λεπτομερής ρύθμιση των ΜΓΜ με αποδείξεις και CoT κάτω από πλαίσια βασισμένα σε κανόνες αποδίδει μερικά από τα καλύτερα αποτελέσματα.

Παζλ Μη-Βασιζόμενα σε Κανόνες Η μελέτη των Lin et al. (2021) [67] καταδεικνύει ότι μοντέλα όπως το BERT [24], RoBERTa [73] και ALBERT [61] αποδίδουν καλύτερα όταν εκπαιδεύονται και στα δύο σύνολα δεδομένων RiddleSense [67] και CommonsenseQA [114], αξιοποιώντας αποτελεσματικά την κοινή γνώση από το δεύτερο σύνολο. Επιπλέον, οι Zhang et al. (2021) [139] αναφέρουν ότι ο συνδυασμός της λεπτομερούς ρύθμισης στο ALBERT-XXL με τη μεταφορά μάθησης από το CommonsenseQA πέτυχε την υψηλότερη ακρίβεια, σημειώνοντας βελτίωση 4% σε σχέση με την απλή λεπτομερή ρύθμιση. Τέλος, η αποτελεσματικότητα της λεπτομερούς ρύθμισης επεκτείνεται στα παζλ κοινής λογικής [23] και στα παζλ προγραμματισμού [107], αναδεικνύοντας την ευρεία εφαρμογή της σε όλες τις κατηγορίες παζλ.

1.2.3 Σύνολα Δεδομένων σε Παζλ

Ντετερμινιστικά Παζλ Βασιζόμενα σε Κανόνες

Το **Sudoku** χρησιμεύει ως ένα βασικό πρόβλημα για τα ΜΓΜ λόγω της λογικής πολυπλοκότητάς του. Οι Noever et al. (2021) [87] έκαναν fine-tuning στο GPT-2 [100] σε 1 εκατομμύριο παιχνίδια Sudoku, πειραματιζόμενοι με μια αναπαράσταση όλου του παζλ με μία μόνο συμβολοσειρά, με τα κενά κελιά να αναπαριστώνται με «-», και διατύπωσε την άποψη ότι μια αναπαράσταση πίνακα μπορεί να ενίσχυε την αποτελεσματικότητα μάθησης του μοντέλου. Στην δουλειά που παρουσιάστηκε από τον Long (2023) [74], χρησιμοποιούνται εμφωλευμένες λίστες για την αναπαράσταση παζλ[1], βρίσκοντας τη μέθοδο Tree-of-Thought (ToT) πιο αποτελεσματική, ειδικά για μικρότερα παζλ. Οι Ishay et al. (2023) [48] διερευνούν νευροσυμβολικές προσεγγίσεις σε Sudoku, Jobs Puzzles και λογικούς γρίφους, αποδεικνύοντας ότι τα ΜΓΜ με καλές προτροπές μπορούν να παράγουν με ακρίβεια answer set programming (ASP) κανόνες.

Για τον **Κύβο του Rubik** και τους **Λαβύρινθους**, έχει αξιολογηθεί η συλλογιστική αντίληψης χώρου του GPT-2 χρησιμοποιώντας πάνω από 2.400 δείγματα Κύβων του Rubik και 10 χιλιάδες λαβύρινθους [87]. Παρά την περιορισμένο χρόνο fine-tuning και τον περιορισμένο αριθμό συμβόλων (tokens), το GPT-2 έλυσε με επιτυχία τον κύβο του Rubik σε 1 από τις 7 προσπάθειες, δείχνοντας δυνατότητες επίλυσης, παρά το υψηλό ποσοστό έγκυρων αν και λανθασμένων λύσεων και το μέγεθος του μοντέλου. Οι Ding et al. (2023) [25] εφαρμόζουν πολλαπλές μεθόδους όπως CoT, Self-Consistency και διάφορες "Δομές Σκέψεις" (ToT, GoT, XoT) σε έναν κύβο του Rubik 2×2×2 χρησιμοποιώντας GPT-3.5 και GPT-4. Το XoT με self-consistency αναδεικνύεται ως η πιο ακριβής μέθοδος, ξεπερνώντας σημαντικά τις άλλες με ποσοστό επιτυχίας 77.6%.

Εξερευνώντας την ευελιξία των ΜΓΜ, έχει αξιολογηθεί η αποτελεσματικότητα του XoT στο πρόβλημα αντίληψης χώρου **8-Puzzle** και στο αριθμητικό παζλ **Game of 24** [25]. Τα προβλήματα του 8-Puzzle επιλύονται με αξιοσημείωτη ακρίβεια 93.2% σε 419 παζλ χρησιμοποιώντας το XoT με αναθεώρηση, παρουσιάζοντας υψηλότερη αποτελεσματικότητα σε σχέση με την προτροπή λίγων παραδειγμάτων και το CoT. Αυτή η υψηλή ακρίβεια, σε συνδυασμό με ένα μειωμένο αριθμό κλήσεων ΜΓΜ, υπογραμμίζει την αποτελεσματικότητα και τις δυνατότητες του XoT σε σύνθετα πλαίσια επίλυσης παζλ.

[1]π.χ. [[3,*,*,*,2], [1,*,3,*],[*,1,*,3],[4,*,*,*,1]]

Σχετικά με τα **Σταυρόλεξα / Κρυπτόλεξα**, υπάρχουν δουλειές [104, 28] που εφαρμόζουν fine-tuning στα μοντέλα T5 [101] σε εκτεταμένα σύνολα δεδομένων μεμονωμένων γρίφων κρυπτόλεξων, αποκαλύπτοντας το πλεονέκτημα του T5, που βασίζεται σε Transformers, έναντι των παραδοσιακών μεθόδων και αναδεικνύοντας περιοχές για βελτίωση, ιδιαίτερα με υπαινιγμούς, επιπλέον στοιχεία και καθορισμένα μήκη απαντήσεων. Η σύγκριση του BART [64] με το T5 υποδεικνύει ακρίβεια κάτω του 30%, με την παραγωγή εμπλουτισμένη με ανάκτηση (Retrieval Augmented Generation – RAG) να ξεπερνούν τα ΜΓΜ που έχουν υποστεί fine-tuning [59]. Επιπλέον, οι Yao et al. (2023) [134] εφαρμόζουν προτροπή 5 παραδειγμάτων και ToT στο GPT-4 σε σταυρόλεξα βελτιώνοντας σημαντικά την απόδοση επιλύοντας 4 από τους 20 γρίφους και επιτυγχάνοντας ποσοστό επιτυχίας 60% σε επίπεδο λέξης.

Σχετικά με το Σκάκι, υπάρχουν δύο fine-tuned μοντέλα, τα «ChessGPT» και «ChessCLIP», χρησιμοποιώντας μια συλλογή 3.2 εκατομμυρίων **παζλ σε σκάκι** από το σύνολο δεδομένων Lichess[2]. Κάθε παζλ στο σύνολο δεδομένων περιλαμβάνει τη δυσκολία, το θέμα και τη λύση του [30].

Τέλος, οι Kazemi et al. (2023) [51] εισάγουν το **BoardgameQA**, ένα σύνολο δεδομένων που περιλαμβάνει ερωτήσεις πολλαπλών επιλογών με αντιφατικά γεγονότα και κανόνες. Τα μοντέλα θα πρέπει να αντιμετωπίσουν αυτές τις πολυπλοκότητες για να παρέχουν απαντήσεις σε μορφή ελεύθερου κειμένου. Η αξιολόγησή τους αποκαλύπτει ότι τα BERT-large και T5-XXL με περαιτέρω ρύθμιση του μοντέλου με αποδείξεις (fine-tuning with proofs) αναδεικνύεται ως η πιο αποτελεσματική μέθοδος, σε αντίθεση με την προτροπή λίγων παραδειγμάτων με CoT στο PaLM. Επίσης, η παρουσία επιπλέον ή αντικρουόμενων πληροφοριών μειώνει την ακρίβεια.

Στοχαστικά Παζλ Βασιζόμενα σε Κανόνες

Το **BoardgameQA** [51] διερευνά επίσης σενάρια με ελλιπείς πληροφορίες, τα οποία εμπίπτουν στην κατηγορία των στοχαστικών παζλ. Αποδεικνύεται ότι όσο αυξάνονται οι ελλείπουσες πληροφορίες, μειώνεται η ακρίβεια των μοντέλων που έχουν υποστεί fine-tuning. Ωστόσο, αυτή η αυξημένη δυσκολία δεν επηρεάζει με τον ίδιο τρόπο την απόδοση των μεθόδων μάθησης περαιτέρω ρύθμισης με προτροπές και προτροπές λίγων παραδειγμάτων, γεγονός που πιθανότατα οφείλεται στα μεγαλύτερα μοντέλα που εφαρμόστηκαν.

Το παιχνίδι του **Ναρκαλιευτή**, γνωστό για την κρυφή πληροφορία που διαθέτει, αποτελεί παράδειγμα στοχαστικών παζλ, απαιτώντας από τους παίκτες να συμπεράνουν τις θέσεις των ναρκών από αριθμητικές ενδείξεις. Οι Li et al. (2023) [66] αξιολόγησαν ΜΓΜ στο Ναρκαλιευτή, συγκρίνοντας αναπαραστάσεις των πλακιδίων του παιχνιδιού με μορφή πίνακα αλλά και συντεταγμένων. Παρόλο που το GPT-3.5 εμφάνισε αρχική κατανόηση, οι μέθοδοι όπως η προτροπή λίγων παραδειγμάτων έδειξαν ελάχιστες βελτιώσεις. Αντίθετα, το GPT-4 βελτίωσε τον εντοπισμό ναρκών, αλλά δυσκολεύτηκε να ολοκληρώσει ένα ολόκληρο πίνακα παχνιδιού. Με βάση τα πειράματα που διεξήχθησαν, η αναπαράσταση συντεταγμένων βοήθησε στην καλύτερη κατανόηση των ΜΓΜ έναντι της μορφής πίνακα.

Τα παιχνίδια καρτών, κυρίως το πόκερ, αποτελούν παράδειγμα στοχαστικών γρίφων όπου η στρατηγική ικανότητα είναι ζωτικής σημασίας. Οι απλοποιημένες παραλλαγές πόκερ απαιτούν από τους παίκτες να συμπεράνουν τις κάρτες των αντιπάλων και να υπολογίσουν τις πιθανότητες εν μέσω κρυφών προθέσεων. Οι Gupta et al. (2023) [42] διαπίστωσαν ότι στον γύρο πριν από το flop του πόκερ, τα ChatGPT και GPT-4 κατανοούν προηγμένες στρατηγικές αλλά δεν φτάνουν στο βέλτιστο παιχνίδι της Θεωρίας Παιγνίων (GTO). Το ChatGPT κλίνει προς μια συντηρητική προσέγγιση, ενώ το GPT-4 παρουσιάζει πιο επιθετικό παιχνίδι. Οι Huang et al. (2024) [44] αξιοποιούν ένα μοντέλο OPT-1.3B που εκπαιδεύτηκε με χρήση την Ενισχυτικής Μάθησης σε όλες τις φάσεις του πόκερ, αποκαλύπτοντας ανώτερα αποτελέσματα σε ποσοστά νίκης και αποδοτικότητας, αναδεικνύοντας τελικά την ικανότητα των ΜΓΜ σε σύνθετες στρατηγικές σε στοχαστικά περιβάλλοντα. Ένας πράκτορας που αξιοποιεί το GPT-4 [41] επιτυγχάνει επίσης σημαντικά αποτελέσματα σε διάφορα παιχνίδια καρτών με ελλιπή πληροφόρηση.

Τα **παιχνίδια κοινωνικής επαγωγής**, συμπεριλαμβανομένων των Werewolf και Avalon, συνδυάζουν τη λογική συλλογιστική με πολύπλοκες κοινωνικές αλληλεπιδράσεις, καθιστώντας τα μέρος του ευρύτερου τομέα των στοχαστικών γρίφων. Τέτοια παιχνίδια προκαλούν τους παίκτες να συμπεράνουν ρόλους που περιλαμβάνουν απρόβλεπτη ανθρώπινη συμπεριφορά. Οι Xu et al. (2023) [131] προτείνουν ένα πλαίσιο για το Werewolf που χρησιμοποιεί ΜΓΜ, αξιοποιώντας παλιότερες αλληλεπιδράσεις για στρατηγικές αποφάσεις και αναδεικνύοντας την ικανότητα των μοντέλων σε αυτό το πλαίσιο. Ομοίως, τα πλαίσια για το Avalon [121, 60] δείχνουν πώς τα ΜΓΜ μπορούν να πλοηγηθούν σε σενάρια που απαιτούν κοινωνικό χειρισμό και εξαγωγή συμπερασμάτων,

[2]https://lichess.org/

υπογραμμίζοντας την ικανότητα των ΜΓΜ στη διαχείριση της πολύπλοκης αλληλεπίδρασης της λογικής και της κοινωνικής αλληλεπίδρασης που είναι εγγενής σε τέτοια παιχνίδια.

Γρίφοι Μη-Βασιζόμενοι σε Κανόνες

Το **RiddleSense** [67] προσφέρει μια συλλογή από 5.7 χιλιάδες αινίγματα γραμμικής, σειριακής σκέψης, δοκιμάζοντας προ-εκπαιδευμένα γλωσσικά μοντέλα όπως τα BERT, RoBERTa, ALBERT και μοντέλα QA από κείμενο σε κείμενο, συμπεριλαμβανομένων των UnifiedQA [52] και T5. Τα μεγαλύτερα μοντέλα επιδεικνύουν γενικά καλύτερες επιδόσεις, με το UnifiedQA που χρησιμοποιεί το T5-3B να προηγείται, αλλά να δυσκολεύεται με τις μεταφορές και τις αντιφατικές καταστάσεις.

Το **BrainTeaser** [50] εισάγει 1119 γρίφους δημιουργικής, αντισυμβατικής σκέψης. Αντιπαραβάλλει μοντέλα που έχουν υποστεί fine-tuning σε οδηγίες (instruction-based fine-tuning) (ChatGPT, T0 και FlanT5 [19]) με μοντέλα που έχουν εκπαιδευτεί σε σύνολα δεδομένων κοινής λογικής (συμπεριλαμβανομένων των παραλλαγών RoBERTa και CAR [122]). Το ChatGPT υπερέχει τόσο σε γρίφους που βασίζονται σε προτάσεις όσο και σε γρίφους που βασίζονται σε λέξεις, υποδεικνύοντας τη δύναμή του στην δημιουργική σκέψη. Ωστόσο, συνολικά, τα ΜΓΜ εξακολουθούν να αντιμετωπίζουν προκλήσεις στην επίδειξη δημιουργικής σκέψης, με κοινά λάθη στην απομνημόνευση και τη συσχέτιση κοινών εννοιών. Αυτό το σύνολο δεδομένων αναδεικνύει τις ποικίλες διαστάσεις της συλλογιστικής που μπορούν να δοκιμάσουν οι γρίφοι, από την γραμμική λογική έως την αντισυμβατική συμπερασματολογία.

Το **BiRdQA** [139] διερευνά την πολύγλωσση πτυχή των γρίφων, περιλαμβάνοντας αγγλικούς και κινεζικούς γρίφους, ενώ αξιολογεί μονόγλωσσα (BERT, RoBERTa), καθώς και πολύγλωσσα (mBERT, XLM-R [21]) γλωσσικά μοντέλα. Δοκιμάζεται επίσης η χρήση σύντομων εισαγωγών και υποδείξεων γρίφων. Τα ευρήματα αποκαλύπτουν ένα σημαντικό χάσμα επιδόσεων μεταξύ των γλωσσικών μοντέλων και της κατανόησης σε ανθρώπινο επίπεδο, με τα μονόγλωσσα μοντέλα να υπερτερούν γενικά έναντι των πολύγλωσσων. Είναι ενδιαφέρον ότι το πρόσθετο περιεχόμενο, όπως οι εισαγωγές και οι υποδείξεις με βάση πληροφορίες από την Wikipedia, ποικίλλει ως προς την αποτελεσματικότητα, με τέτοια βοηθήματα να ωφελούν τους αγγλικούς αλλά όχι τους κινεζικούς γρίφους.

Το **CC-Riddle** επικεντρώνεται σε 27 χιλιάδες αινίγματα κινεζικών χαρακτήρων, που περιλαμβάνουν μορφές πολλαπλής επιλογής [128]. Η αξιολόγηση καταδεικνύει ότι τα μοντέλα αντιμετώπισαν δυσκολίες στην κατανόηση και παρουσίασαν παρανοήσεις, αποκαλύπτοντας την πολυπλοκότητα που μπορεί να υπάρχει στα αινίγματα με βάση τους χαρακτήρες.

Το **PUZZLEQA** [142] προσφέρει 558 γρίφους λέξεων σε μορφή πολλαπλής επιλογής και ελεύθερου κειμένου. Τα μεγαλύτερα μοντέλα, π.χ. GPT-3/3.5, παρουσιάζουν μεγαλύτερη ακρίβεια, ειδικά σε προβλήματα πολλαπλής επιλογής. Ωστόσο, μέθοδοι όπως το CoT σε συνδυασμό με την περίληψη δεν βελτιώνουν σημαντικά τις επιδόσεις, υποδεικνύοντας τις συνεχιζόμενες προκλήσεις στην επίλυση γρίφων ελεύθερης απάντησης.

Τέλος, το **MARB** [116] περιλαμβάνει μια ποικιλία από γρίφους. Διάφορες μεθοδολογίες, συμπεριλαμβανομένων των CoT, IEP (Inferencial Exclusion Prompting) και προτροπή μηδενικών ή μερικών παραδειγμάτων, δοκιμάζονται σε μοντέλα όπως το GPT-4 και το PaLM2-540B [2]. Ο συνδυασμός IEP και CoT αναδείχθηκε ως η πιο αποτελεσματική μέθοδος, αναδεικνύοντας την αξία της ενσωμάτωσης πολλαπλών προσεγγίσεων για διάφορους τύπους αινιγμάτων. Το σύνολο δεδομένων περιλαμβάνει επίσης παζλ κοινής λογικής, παρουσιάζοντας παρόμοιες τάσεις με τους γρίφους.

Προγραμματιστικά Παζλ Μη-Βασιζόμενα σε Κανόνες

Το **P3 (Python Programming Puzzles)** [107] προσφέρει μια σειρά από προκλήσεις προγραμματισμού Python, από απλούς χειρισμούς συμβολοσειρών μέχρι σύνθετες εργασίες, όπως ο Πύργος του Ανόι και αλγοριθμικά παζλ, που απαιτούν από το μοντέλο να βρει μια είσοδο που κάνει το πρόγραμμα f να επιστρέψει «True». Τα μοντέλα που εφαρμόζονται σε αυτούς τους γρίφους περιλαμβάνουν κλασσικούς επιλυτές για τη δημιουργία Αφηρημένων Συντακτικών Δένδρων (Abstract Syntax Trees) και γλωσσικά μοντέλα, όπως το GPT-3 και το Codex [15], χρησιμοποιώντας ποικίλες τεχνικές προτροπής. Η μετρική αξιολόγησης pass@k, δείχνει την ικανότητα των μοντέλων να επιλύουν ένα γρίφο μέσα σε ένα συγκεκριμένο αριθμό προσπαθειών [15]. Τα αποτελέσματα δείχνουν συσχέτιση μεταξύ της δυσκολίας του γρίφου τόσο για τα μοντέλα όσο και για τους ανθρώπους, με τις περιγραφικές υποδείξεις να ενισχύουν την απόδοση των μοντέλων. Είναι ενδιαφέρον ότι τα μοντέλα που ήταν ικανά στη συμπλήρωση κώδικα έλυσαν περισσότερους γρίφους με λιγότερες προσπάθειες, υπογραμμίζοντας τη σημασία των εξειδικευμένων ικανοτήτων στις προγραμματιστικές προκλήσεις.

Οι Savelka et al. (2023) [106] παρουσιάζουν ένα σύνολο δεδομένων που αποτελείται από 530 αποσπάσματα κώδικα από μαθήματα προγραμματισμού, παρουσιάζοντας γρίφους σε μορφή πολλαπλών επιλογών. Η διάκριση μεταξύ ερωτήσεων με και χωρίς αποσπάσματα κώδικα προσφέρει μια διαφορετική προσέγγιση για τις στρατηγικές επίλυσης προβλημάτων των ΜΓΜ. Το σύνολο δεδομένων κατηγοριοποιεί τις ερωτήσεις σε έξι τύπους, συμπεριλαμβανομένων των ερωτήσεων σωστού/λάθους και πρόβλεψης εξόδου. Τα μοντέλα GPT αξιολογήθηκαν, αποκαλύπτοντας ότι η συμπερίληψη κώδικα αυξάνει σημαντικά την πολυπλοκότητα του γρίφου. Τα ποσοστά ακρίβειας ποικίλλουν, με υψηλότερες επιδόσεις σε ερωτήσεις προσανατολισμένες στην ολοκλήρωση, γεγονός που υποδηλώνει ότι η αποτελεσματικότητα των ΜΓΜ μπορεί να εξαρτάται σε μεγάλο βαθμό από τη μορφή και το περιεχόμενο των ερωτήσεων.

Ενώ τόσο το P3 όσο και το Programming Snippets Dataset αντιμετωπίζουν γρίφους προγραμματισμού, το κάνουν με σημαντικά διαφορετικούς τρόπους. Η εστίαση του P3 στην εύρεση σωστών εισόδων προγράμματος Python έρχεται σε αντίθεση με τη μορφή πολλαπλών επιλογών του Programming Snippets Dataset. Ωστόσο, και τα δύο σύνολα δεδομένων αποκαλύπτουν βασικά συμπεράσματα: οι περιγραφικές υποδείξεις βοηθούν στην επίλυση προβλημάτων και η μορφή των ερωτήσεων επηρεάζει σημαντικά την απόδοση των ΜΓΜ.

Παζλ Κοινής Λογικής Μη-Βασιζόμενα σε Κανόνες

Το **True Detective** [23] παρουσιάζει παζλ τύπου ντετέκτιβ σε ιστορίες μεγάλου μήκους, προκαλώντας τα ΜΓΜ όπως το GPT-3.5/4 να βγάλουν συμπεράσματα. Εφαρμόζονται διάφορες μέθοδοι, συμπεριλαμβανομένων των CoT και Golden-CoT, αποκαλύπτοντας δυσκολίες στην εξαγωγή τελικών συμπερασμάτων παρά το γεγονός ότι όλες οι απαραίτητες πληροφορίες είναι διαθέσιμες. Το Golden-CoT παρέχει στο μοντέλο το σκεπτικό πίσω από τη σωστή απάντηση, οπότε το μοντέλο χρειάζεται μόνο να κατανοήσει αυτό το σκεπτικό και να εξάγει την απάντηση. Ενώ οι προσεγγίσεις Vanilla και CoT έχουν σχεδόν τυχαίες επιδόσεις, η Golden-CoT επιδεικνύει σημαντικά καλύτερη ακρίβεια, ιδίως με το GPT-4. Ωστόσο, ακόμη και με το Golden-CoT, το GPT-3.5 επιτυγχάνει ποσοστό επίλυσης μόνο 63%, ενώ το GPT-4 αντιστοιχεί στα αποτελέσματα των ανθρώπινων λύσεων (χωρίς πρόσβαση στη συλλογιστική πίσω από την απάντηση).

Το **DetectBench** [40] που περιέχει 1200 ερωτήσεις, αξιολογεί επίσης την συλλογιστική σε πραγματικές συνθήκες. Ελέγχει μεθόδους όπως η χρήση υποδείξεων, διάφορες παραλλαγές του CoT και το Detective Thinking σε μοντέλα όπως τα GPT-4, GPT-3.5, GLM-4 και Llama2. Οι υποδείξεις φαίνεται πως αποτελούν ένα ισχυρό βοήθημα, με τα μεγαλύτερα μοντέλα να υπερτερούν γενικά των μικρότερων. Η αποτελεσματικότητα των διαφόρων προσεγγίσεων ποικίλλει, με το Detective Thinking να βοηθά αποτελεσματικά τα περισσότερα μοντέλα.

Το **LatEval** [47] εισάγει μια μορφή συνομιλίας με αγγλικές και κινεζικές ιστορίες, απαιτώντας από τους παίκτες να κάνουν ερωτήσεις ναι/όχι πριν δώσουν μια απάντηση. Το GPT-3.5, το GPT-4 και διάφορα άλλα μοντέλα συνομιλίας αξιολογούνται ως προς την ικανότητά τους να θέτουν σχετικές ερωτήσεις και να διατηρούν τη συνέπεια με την αλήθεια. Τα μεγαλύτερα μοντέλα δεν παρουσιάζουν απαραίτητα προηγμένες επιδόσεις στη συνάφεια των ερωτήσεων. Ωστόσο, το GPT-4 επιδεικνύει την υψηλότερη συνέπεια απαντήσεων, αν και υπάρχουν ακόμη σημαντικά περιθώρια βελτίωσης. Το σύνολο δεδομένων υπογραμμίζει τη σημασία της διαδραστικής και συνομιλιακής συλλογιστικής στην κατανόηση της κοινής λογικής.

1.3 Πειράματα

Στην ενότητα αυτή εξετάζουμε τις επιδόσεις μικρότερων μοντέλων, όπως τα Llama2-7B, Llama3-8B, Llama3.1-8B, Llama3.1-70B και Mistral-7B, (όλα τα μοντέλα είναι instruction fine-tuned, δηλαδή έχουν υποστεί λεπτομερή ρύθμιση των παραμέτρων τους για να αντιλαμβάνονται οδηγίες που τους δίνονται) τόσο σε εργασίες μαθηματικού συλλογισμού όσο και σε εργασίες επίλυσης παζλ. Προηγούμενες έρευνες έχουν καταδείξει την αποτελεσματικότητα προηγμένων τεχνικών προτροπής, όπως τα αλυσιδωτά βήματα σκέψης (Chain-of-Thought), οι νευροσυμβολικές μέθοδοι και η προτροπή από το λιγότερο στο περισσότερο σύνθετο υποπρόβλημα (least-to-most – LtM) σε μεγαλύτερα μοντέλα (π.χ. μοντέλα Codex ή GPT-4). Ωστόσο, η εφαρμογή αυτών των τεχνικών σε μικρότερα μοντέλα παραμένει ανεξερεύνητη, ιδίως σε τομείς που απαιτούν λογική συλλογιστική και επίλυση προβλημάτων σε πολλαπλά βήματα υπό περιορισμένους υπολογιστικούς πόρους.

Σε αυτή την ενότητα, εστιάζουμε σε τέσσερα σύνολα δεδομένων: δύο σύνολα δεδομένων μαθηματικής συλλογιστικής - *GSM8K* και *SVAMP* - και δύο σύνολα δεδομένων συλλογιστικής παζλ, *RiddleSense* και *Game-of-24*. Εφαρμόζουμε διάφορες τεχνικές προτροπής (IO prompting, CoT, LtM και Faithful-CoT / PoT) υπό διάφορες ρυθμίσεις, συμπεριλαμβανομένης της προτροπής με μηδενικό αριθμό παραδειγμάτων, με λίγα παραδείγματα και με αυτοσυνέπεια. Αυτές οι τεχνικές, ιδίως οι νευροσυμβολικές, έχουν δείξει πολλά υποσχόμενα αποτελέσματα με μεγαλύτερα μοντέλα, αλλά η αποτελεσματικότητά τους με μικρότερα, πιο αποδοτικά σε πόρους μοντέλα δεν έχει ακόμη κατανοηθεί πλήρως. Επιπλέον, οι νευροσυμβολικές μέθοδοι -όπου τα προβλήματα μετατρέπονται σε κώδικα για εξωτερικές μηχανές συλλογισμού- έχουν σπάνια δοκιμαστεί για την επίλυση παζλ, προσφέροντας μια καινοτόμο κατεύθυνση που διερευνάται στην παρούσα μελέτη.

1.3.1 Σύνολα Δεδομένων

GSM8K (Grade School Math 8K)

Το GSM8K [20] είναι ένα σύνολο δεδομένων που έχει σχεδιαστεί για την αξιολόγηση των ικανοτήτων μαθηματικής συλλογιστικής μεγάλων γλωσσικών μοντέλων. Περιλαμβάνει 8.5 χιλιάδες μαθηματικά προβλήματα, χωρισμένα σε 7.5 χιλ. προβλήματα εκπαίδευσης και 1000 προβλήματα δοκιμής, με τα πειράματά μας να διεξάγονται στο σύνολο δοκιμής 1000 προβλημάτων. Κάθε πρόβλημα απαιτεί 2 έως 8 λογικά βήματα και περιλαμβάνει βασικές αριθμητικές πράξεις, καθιστώντας το κατάλληλο για την αξιολόγηση των ικανοτήτων επίλυσης προβλημάτων των ΜΓΜ σε επίπεδο δυσκολίας προσιτό για μαθητές γυμνασίου.

SVAMP (Simple Variations on Arithmetic Math Word Problems)

Το SVAMP [94] είναι ένα σύνολο δεδομένων που αποσκοπεί στον έλεγχο της ικανότητας των μοντέλων στην επίλυση μαθηματικών λεκτικών προβλημάτων στοιχειώδους επιπέδου (Math Word Problems – MWPs). Αποτελείται από *1000 αριθμητικά προβλήματα* που έχουν σχεδιαστεί για να προκαλέσουν τις ικανότητες συλλογισμού των μοντέλων μέσω προβλημάτων που επιλύονται με έως και δύο αριθμητικές πράξεις (π.χ. πρόσθεση, αφαίρεση, πολλαπλασιασμός, διαίρεση), κατάλληλα για μαθητές μικρών τάξεων.

RiddleSense

Το RiddleSense [67] είναι ένα σύνολο δεδομένων που αποσκοπεί στην αξιολόγηση των ικανοτήτων των γλωσσικών μοντέλων να λύνουν γρίφους μη βασιζόμενους σε κανόνες που απαιτούν εναλλακτική σκέψη, γλωσσική κατανόηση και κοινή λογική. Αποτελούμενο από 5.7 χιλ. γρίφους πολλαπλής επιλογής, χρησιμοποιήσαμε ένα υποσύνολο 3 χιλ. τυχαία επιλεγμένων γρίφων για τα πειράματά μας. Κάθε γρίφος, που παρουσιάζεται σε φυσική γλώσσα, απαιτεί από τα μοντέλα να αξιοποιήσουν τη γλωσσική γνώση και την κατανόηση του πραγματικού κόσμου για να αποκωδικοποιήσουν κρυμμένα νοήματα και μεταφορικές ενδείξεις (π.χ. «Τι γίνεται πιο υγρό όσο περισσότερο στεγνώνει;» με την απάντηση «μια πετσέτα»).

Game-of-24

Το σύνολο δεδομένων με παζλ Game of 24 [134] έχει σχεδιαστεί για να αξιολογεί τη μαθηματική λογική, προκαλώντας τα μοντέλα να χειριστούν τέσσερις αριθμούς (που κυμαίνονται από το 1 έως το 13) χρησιμοποιώντας βασικές αριθμητικές πράξεις (πρόσθεση, αφαίρεση, πολλαπλασιασμό, διαίρεση) για να φτάσουν στο αποτέλεσμα 24. Κάθε πρόβλημα βασίζεται σε κανόνες και είναι ντετερμινιστικό, απαιτώντας λογική σκέψη εντός αυστηρών μαθηματικών περιορισμών. Το σύνολο δεδομένων περιλαμβάνει 1362 προβλήματα, με 1225 για την εκπαίδευση και 137 για δοκιμή. Τα τελευταία χρησιμοποιήθηκαν για την αξιολόγηση των πειραμάτων μας. Το σύνολο δοκιμών, που περιλαμβάνει προβλήματα ταξινομημένα με βάση το χρόνο επίλυσης από τον άνθρωπο για να υποδείξει τη δυσκολία, απαιτεί από τα μοντέλα να εφαρμόζουν στρατηγική σκέψη και συλλογισμό πολλών βημάτων για να εξερευνήσουν διάφορους συνδυασμούς πράξεων και αριθμών. Αυτή το πρόβλημα είναι πιο σύνθετο από τα βασικά μαθηματικά προβλήματα, καθώς δοκιμάζει όχι μόνο την απλή αριθμητική, αλλά και την ικανότητα ενός μοντέλου για συλλογισμό δοκιμής και λάθους και εξερεύνηση στρατηγικών λύσεων.

1.3.2 Γλωσσικά Μοντέλα

Στην παρούσα διπλωματική εργασία, δίνουμε προτεραιότητα στη χρήση *μικρότερης κλίμακας ΜΓΜ*, που κυμαίνονται από 7 δισεκατομμύρια έως 70 δισεκατομμύρια παραμέτρους. Ενώ οι πρόσφατες πρόοδοι στην ανάπτυξη

ΜΓΜ έχουν επικεντρωθεί γύρω από μοντέλα με εκατοντάδες δισεκατομμύρια παραμέτρους, υπάρχει αυξανόμενο ενδιαφέρον για τη διερεύνηση του πόσο καλά μπορούν να αποδώσουν μικρότερα μοντέλα σε πολύπλοκες εργασίες συλλογισμού. Τα μικρότερα μοντέλα, όπως τα Llama2-7b και Mistral-7b, είναι ταχύτερα και πιο αποδοτικά από υπολογιστική άποψη, καθιστώντας τα πιο πρακτικά για ένα ευρύτερο φάσμα εφαρμογών. Επιπλέον, η δοκιμή προηγμένων τεχνικών προτροπής, όπως το Chain-of-Thought και το Faithful-CoT, με αυτά τα μικρότερα μοντέλα μπορεί να αποκαλύψει κατά πόσο τέτοιες μέθοδοι μπορούν να βοηθήσουν στη γεφύρωση του χάσματος μεταξύ μικρότερων και μεγαλύτερων μοντέλων, ξεκλειδώνοντας ενδεχομένως αναδυόμενες ικανότητες συλλογισμού χωρίς την ανάγκη για τεράστιο αριθμό παραμέτρων. Βασιζόμαστε σε προ-εκπαιδευμένα μοντέλα από το Hugging Face, επιλέγοντας συγκεκριμένα παραλλαγές συντονισμένες με οδηγίες, σχεδιασμένες να ακολουθούν πιο αποτελεσματικά τις οδηγίες φυσικής γλώσσας.

1.3.3 Μεθοδολογίες

Χρησιμοποιήσαμε διάφορες τεχνικές προτροπής και μεθόδους συλλογισμού για να αξιολογήσουμε την απόδοση των μικρότερων ΜΓΜ σε εργασίες μαθηματικής συλλογιστικής (GSM8K, SVAMP) και επίλυσης παζλ (Game of 24). Οι μέθοδοι που εφαρμόσαμε περιλαμβάνουν την τυπική προτροπή εισόδου-εξόδου (IO), την προτροπή με αλυσιδωτά βήματα σκέψης (CoT), την προτροπή από το μικρότερο στο μεγαλύτερο υποπρόβλημα (LtM) και τη μετάφραση γρίφων, με την μέθοδο Faithful-CoT για τα σύνολα δεδομένων μαθηματικών και μια εξειδικευμένη προσαρμογή του Faithful-CoT που χρησιμοποιήθηκε για τη μετάφραση φυσικής γλώσσας σε κώδικα Python για το Game of 24.

Τυπική Προτροπή Εισόδου-Εξόδου

Η τυπική προτροπή Ε/Ε περιλαμβάνει την παροχή στο μοντέλο μιας ερώτησης και της αντίστοιχης απάντησης στο περιεχόμενο. Το μοντέλο στη συνέχεια προσπαθεί να αναπαράγει αυτό το μοτίβο για νέες εισόδους. Αυτή η μέθοδος χρησιμεύει ως μια απλή βασική γραμμή συλλογισμού. Τόσο για τα σύνολα δεδομένων μαθηματικών όσο και για τα σύνολα δεδομένων γρίφων, η προτροπή Ε/Ε παρουσιάζει το πρόβλημα σε φυσική γλώσσα και αναμένει από το μοντέλο να εξάγει μια λύση χωρίς να περιγράφει λεπτομερώς τα ενδιάμεσα βήματα συλλογισμού.

Προτροπή με Αλυσιδωτά Βήματα Σκέψης

Η προτροπή CoT ενισχύει τη συλλογιστική ικανότητα των ΜΓΜ αναλύοντας το πρόβλημα σε ενδιάμεσα βήματα. Αντί να βγάζει αμέσως την τελική απάντηση, το μοντέλο προτρέπεται να διατυπώσει τη διαδικασία σκέψης που εμπλέκεται στην επίλυση του προβλήματος. Αυτή η προσέγγιση είναι ιδιαίτερα χρήσιμη για εργασίες που απαιτούν συλλογισμό πολλών βημάτων, καθιστώντας την αποτελεσματική τόσο για μαθηματικά προβλήματα όσο και για γρίφους που περιλαμβάνουν πολύπλοκες πράξεις.

Προτροπή από το Λιγότερο στο Περισσότερο Σύνθετο Υποπρόβλημα

Η προτροπή LtM [143], έχει σχεδιαστεί για να χειρίζεται πολύπλοκα προβλήματα με την αποσύνθεσή τους σε μια σειρά μικρότερων, απλούστερων υποπροβλημάτων. Σε αντίθεση με το CoT, το LtM απλοποιεί ρητά το αρχικό πρόβλημα σε επιμέρους βήματα πριν επιλύσει κάθε μέρος με προοδευτικό τρόπο. Η μέθοδος αυτή έχει αποδειχθεί ιδιαίτερα χρήσιμη στην επίλυση μαθηματικών προβλημάτων, όπου τα πολύπλοκα ερωτήματα μπορούν να αναλυθούν σε βασικές αριθμητικές πράξεις ή λογικά υποστοιχεία.

Faithful-CoT

Το Faithful-CoT [78] είναι μια μέθοδος δύο σταδίων που εισάγεται για να συνδυάζει τη συλλογιστική σε φυσική γλώσσα (ΦΓ) και την εκτέλεση σε συμβολική γλώσσα (ΣΓ). Σε αυτή την προσέγγιση, το μοντέλο παράγει πρώτα μια αλυσίδα συλλογισμού που διαπλέκει περιγραφές φυσικής γλώσσας και συμβολικούς υπολογισμούς (όπως κώδικα Python). Το δεύτερο στάδιο περιλαμβάνει τη χρήση ενός εξωτερικού επιλυτή (π.χ. ενός διερμηνέα Python) για την εκτέλεση του συμβολικού κώδικα και τον υπολογισμό της τελικής απάντησης.

1.3.4 Ρυθμίσεις Προτροπής

Προτροπή Μηδενικών ή Μερικών Παραδειγμάτων

Για όλες τις μεθόδους που εφαρμόστηκαν για τα σύνολα δεδομένων μαθηματικού συλλογισμού, εφαρμόστηκε ρύθμιση 8 παραδειγμάτων, που σημαίνει ότι δόθηκαν 8 (τυχαία) παραδείγματα στο μοντέλο στην προτροπή του χρήστη. Για αυτά τα σύνολα δεδομένων επαναλάβαμε τα πειράματα με και χωρίς τη χρήση μιας προτροπής συστήματος (μια αρχική προτροπή που δίνει στα μοντέλα τις οδηγίες που περιγράφουν τον τρόπο που θέλουμε να απαντήσουν).

Για τα σύνολα δεδομένων γρίφων έχουμε τις ακόλουθες ρυθμίσεις:

- RiddleSense: Τόσο για την τυπική προτροπή Ε/Ε όσο και για το CoT έχουμε δοκιμάσει παραδείγματα με 0, 5, και 8 παραδείγματα.
- Game of 24: Για την τυπική προτροπή Ε/Ε και το CoT έχουμε δοκιμάσει παραδείγματα με μηδενική 0, 5, και 8 παραδείγματα, ενώ για τη μετάφραση γρίφων έχουμε δοκιμάσει μόνο τη ρύθμιση με μηδενικό πλήθος παραδειγμάτων. Για την τελευταία μέθοδο, προσπαθήσαμε επίσης να υποδείξουμε τα βήματα συλλογισμού που πρέπει να ακολουθηθούν.

Αυτοσυνέπεια (Self-Consistency)

Για τα μαθηματικά σύνολα δεδομένων, η αυτοσυνέπεια έχει επίσης δοκιμαστεί για όλες τις μεθόδους, με τη χρήση προτροπής συστήματος. Ο αριθμός των παραγόμενων αλυσίδων για τη μέθοδο της αυτοσυνέπειας είναι n=5, και το πιο συνεπές/ψηφισμένο αποτέλεσμα είναι η τελική απάντηση που δίνεται.

Για το RiddleSense δοκιμάσαμε την αυτοσυνέπεια τόσο για την τυπική προτροπή Ε/Ε όσο και για την CoT και σε κάθε περίπτωση χρησιμοποιήσαμε τη ρύθμιση των μερικών παραδειγμάτων που είχε τα καλύτερα αποτελέσματα στις περισσότερες περιπτώσεις. Για παράδειγμα, για την τυπική προτροπή δοκιμάσαμε την αυτοσυνέπεια στη ρύθμιση 8 παραδειγμάτων, ενώ για το CoT δοκιμάσαμε την αυτοσυνέπεια στη ρύθμιση 5 παραδειγμάτων. Εδώ χρησιμοποιείται ένας αριθμός αλυσίδων ίσος με n=5. Παρομοίως, εφαρμόσαμε αυτοσυνέπεια για το Game of 24, δοκιμάζοντας επίσης έναν αριθμό παραγόμενων αλυσίδων ίσο με n=10.

1.3.5 Αποτελέσματα

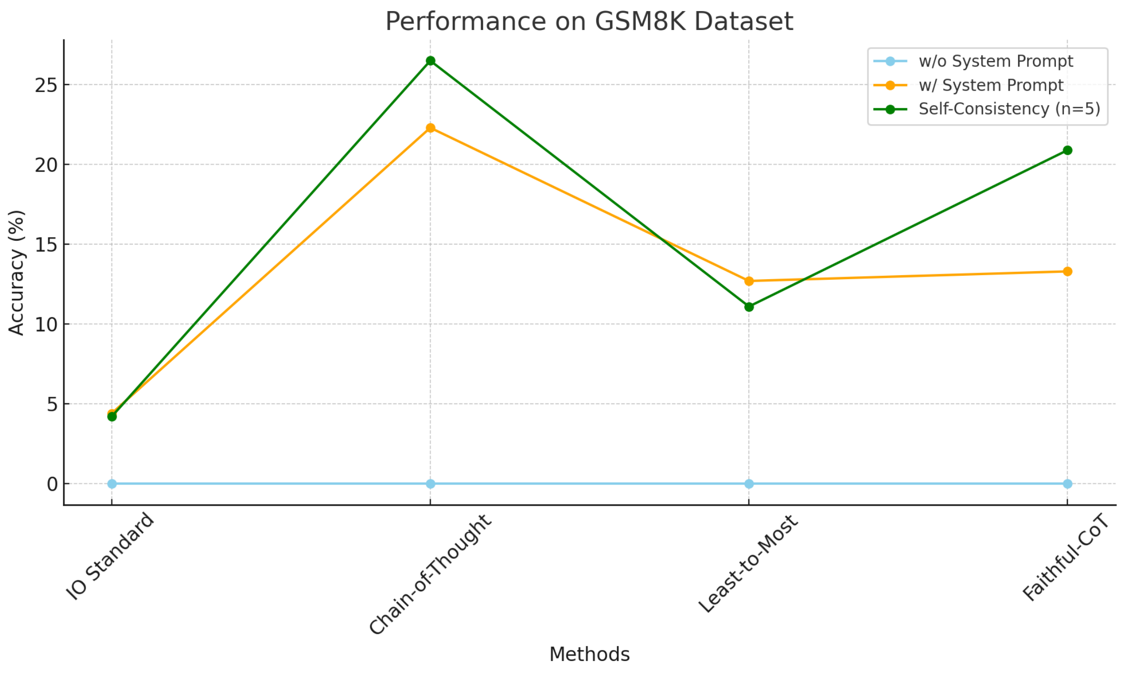
Στην ενότητα αυτή παρουσιάζεται η απόδοση διαφόρων μεθόδων προτροπής και μοντέλων σε τέσσερα σύνολα δεδομένων: GSM8K, SVAMP, RiddleSense και Game of 24. Κάθε σύνολο δεδομένων απεικονίζει μοναδικές πτυχές της συλλογιστικής ικανότητας και των δυνατοτήτων επίλυσης προβλημάτων σε γλωσσικά μοντέλα, συγκρίνοντας τις τυπικές προσεγγίσεις εισόδου-εξόδου (IO), βημάτων αλυσιδωτής σκέψης (CoT), μικρότερων προς μεγαλύτερων σύνθετων υποπροβλημάτων (LtM), Faithful-CoT και μετάφρασης γρίφων σε διαφορετικές ρυθμίσεις προτροπής (ρυθμίσεις μηδενικών ή μερικών παραδειγμάτων και αυτοσυνέπειας).

Αποτελέσματα GSM8K

Τα αποτελέσματα των επιδόσεων για το σύνολο δεδομένων GSM8K παρουσιάζονται παρακάτω:

Methods	**w/o System Prompt**	**w/ System Prompt**	**Self-Consistency (n=5)**
IO Standard	0%	4.4%	4.2%
Chain-of-Thought	0%	22.3%	26.5%
Least-to-Most	0%	12.7%	11.1%
Faithful-CoT	0%	13.3%	20.9%

Table 1.2: Αποτελέσματα GSM8K



Παρατηρήσεις:

- Η απουσία προτροπής συστήματος οδηγεί σε ακρίβεια 0% για όλες τις εφαρμοζόμενες μεθόδους. Με βάση τις παρατηρήσεις μας, αυτό συνέβη λόγω της «υπερεκπαίδευσης» (overfitting) του μοντέλου στα 8 δοσμένα παραδείγματα. Το Llama2-7b-instruct επαναλάμβανε τα παραδείγματα που δόθηκαν αντί να απαντήσει στην τελική ερώτηση.

- Η προσθήκη προτροπής συστήματος βελτίωσε τα αποτελέσματα, αλλά το μοντέλο εξακολουθεί να υπολείπεται στα συγκεκριμένα προβλήματα, με μέγιστη ακρίβεια στη μέθοδο CoT-SC.

- Το CoT αποδείχθηκε η πιο επιτυχημένη μέθοδος, ενώ η αυτοσυνέπεια βελτίωσε τα αποτελέσματα για το CoT και το Faithful-CoT.

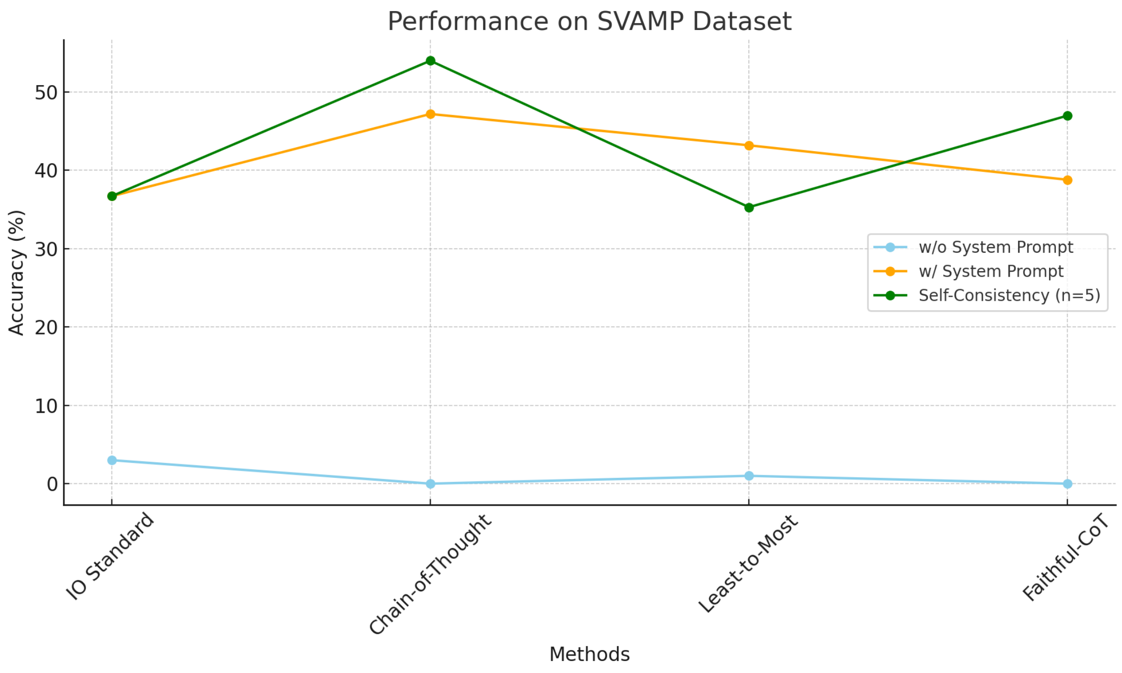
- Σε αντίθεση με την δημοσίευση που εισήγαγε το Faithful-CoT [78], όπου χρησιμοποιήθηκαν μοντέλα όπως τα GPT-3.5 και GPT-4, το CoT πέτυχε καλύτερα αποτελέσματα από τη νευροσυμβολική μέθοδο στα πειράματά μας με το μοντέλο Llama2-7b.

Αποτελέσματα SVAMP

Τα αποτελέσματα των επιδόσεων για το σύνολο δεδομένων SVAMP παρουσιάζονται παρακάτω:

Methods	**w/o System Prompt**	**w/ System Prompt**	**Self-Consistency (n=5)**
IO Standard	3%	36.7%	36.7%
Chain-of-Thought	0%	47.2%	54.0%
Least-to-Most	1%	43.2%	35.3%
Faithful-CoT	0%	38.8%	47.0%

Table 1.3: Αποτελέσματα SVAMP



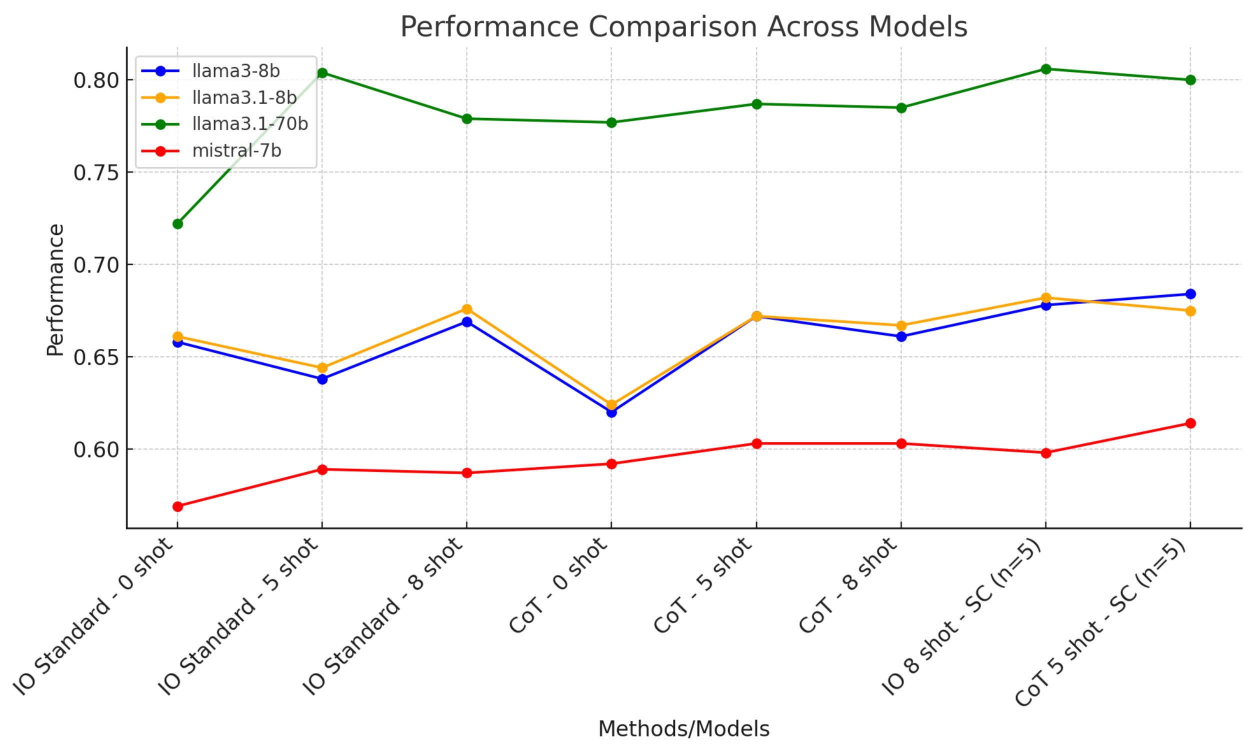
Παρατηρήσεις: Οι παρατηρήσεις για τα αποτελέσματα στο σύνολο δεδομένων SVAMP είναι πολύ παρόμοιες με εκείνες στο GSM8K. Η μόνη διαφορά είναι ότι η ακρίβεια των περισσότερων μεθόδων σε αυτό το σύνολο δεδομένων είναι καλύτερη από τα αποτελέσματα που παρατηρήθηκαν προηγουμένως. Αυτή είναι μια γενική τάση στους διαθέσιμους πίνακες κατάταξης που υπάρχουν και είναι πιθανό να οφείλεται στο γεγονός ότι τα προβλήματα που παρουσιάζονται στο SVAMP είναι ελαφρώς ευκολότερα σε σύγκριση με το GSM8K.

Αποτελέσματα RiddleSense

Τα αποτελέσματα των επιδόσεων για το σύνολο δεδομένων RiddleSense παρουσιάζονται παρακάτω:

Methods/Models	**llama3-8b**	**llama3.1-8b**	**llama3.1-70b**	**mistral-7b**
IO Standard - 0 shot	0.658	0.661	0.722	0.569
IO Standard - 5 shot	0.638	0.644	0.804	0.589
IO Standard - 8 shot	0.669	0.676	0.779	0.587
CoT - 0 shot	0.620	0.624	0.777	0.592
CoT - 5 shot	0.672	0.672	0.787	0.603
CoT - 8 shot	0.661	0.667	0.785	0.603
IO 8 shot - SC (n=5)	0.678	0.682	0.806	0.598
CoT 5 shot - SC (n=5)	0.684	0.675	0.8	0.614

Table 1.4: Αποτελέσματα RiddleSense



Παρατηρήσεις:

- Αρχικά, παρατηρείται ότι ο σημαντικότερος παράγοντας για την επίτευξη καλύτερων αποτελεσμάτων στο RiddleSense είναι το μέγεθος του μοντέλου (ο αριθμός των παραμέτρων του), με το llama3.1-70b-instruct να υπερέχει σταθερά έναντι των άλλων μοντέλων, φτάνοντας μέχρι και 0.806 σε ακρίβεια με τυπική προτροπή 8 παραδειγμάτων με αυτοσυνέπεια. Από την άλλη πλευρά, το mistral-7b-v0.3, αν και αποδοτικό, δυσκολεύεται να φτάσει τις επιδόσεις των μοντέλων llama σε όλες τις περιπτώσεις υπογραμμίζοντας τη σημασία της κλίμακας του μοντέλου για τα αινίγματα.

- Η προτροπή λίγων παραδειγμάτων βελτιώνει τα αποτελέσματα τόσο για την τυπική προτροπή Ε/Ε όσο και για την προτροπή CoT, με τον αριθμό παραδειγμάτων να μην είναι σαφώς καλύτερος μεταξύ 5 και 8.

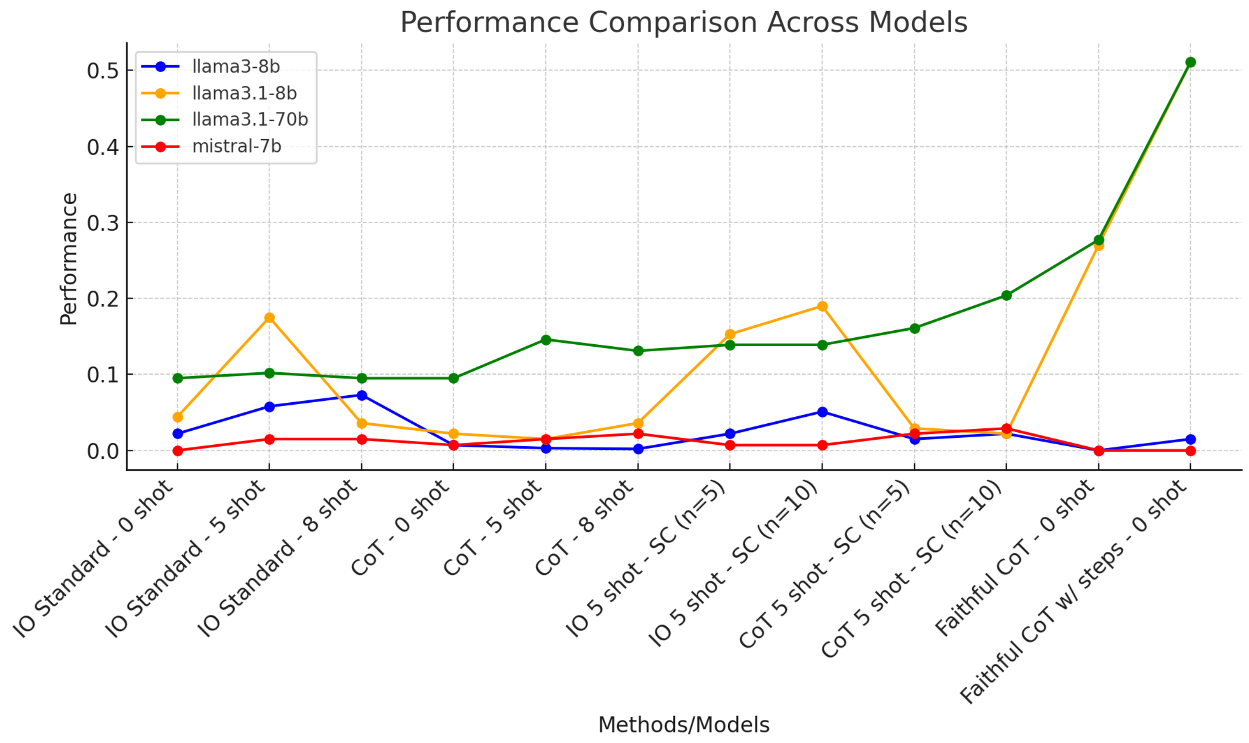
- Η αυτοσυνέπεια (n=5) βελτιώνει γενικά τα αποτελέσματα για όλα τα μοντέλα.

Αποτελέσματα Game-of-24

Τα αποτελέσματα των επιδόσεων για το σύνολο δεδομένων Game-of-24 παρουσιάζονται παρακάτω:

Methods/Models	llama3-8b	llama3.1-8b	llama3.1-70b	mistral-7b
IO Standard - 0 shot	0.022	0.044	0.095	0.0
IO Standard - 5 shot	0.058	0.175	0.102	0.015
IO Standard - 8 shot	0.073	0.036	0.095	0.015
CoT - 0 shot	0.007	0.022	0.095	0.007
CoT - 5 shot	0.003	0.015	0.146	0.015
CoT - 8 shot	0.002	0.036	0.131	0.022
IO 5 shot - SC (n=5)	0.022	0.153	0.139	0.007
IO 5 shot - SC (n=10)	0.051	0.19	0.139	0.007
CoT 5 shot - SC (n=5)	0.015	0.029	0.161	0.022
CoT 5 shot - SC (n=10)	0.022	0.022	0.204	0.029
Faithful CoT - 0 shot	0.0	0.27	0.277	0.0
Faithful CoT w/ steps - 0 shot	0.015	0.511	0.511	0.0

Table 1.5: Αποτελέσματα Game-of-24



Παρατηρήσεις:

- Και πάλι το μέγεθος του μοντέλου παίζει καθοριστικό ρόλο στην απόδοση, με το llama3.1-70b να επιτυγχάνει τα καλύτερα αποτελέσματα για τις περισσότερες περιπτώσεις και το mistral-7b να δυσκολεύεται σε αυτό το περιβάλλον υψηλής πολυπλοκότητας, με τις περισσότερες περιπτώσεις να μην ξεπερνούν την ελάχιστη ακρίβεια.

- Η αύξηση του αριθμού των παραδειγμάτων δεν βελτιώνει σταθερά τα αποτελέσματα για την τυπική προτροπή και το CoT.

- Σε ορισμένες περιπτώσεις, το CoT όχι μόνο δεν βελτιώνει τις επιδόσεις αλλά παράγει ακόμη και χειρότερα αποτελέσματα από την τυπική προτροπή Ε/Ε. Επίσης, η αυτοσυνέπεια βοήθησε ορισμένα μοντέλα να έχουν καλύτερες επιδόσεις, αλλά αυτό δεν συνέβαινε πάντα.

- Όσον αφορά την τεχνική μετάφρασης παζλ, παρατηρείται ότι τα mistral-7b και llama3-8b δεν είναι σε θέση να δημιουργήσουν εκτελέσιμο κώδικα, οδηγώντας έτσι σε αποτελέσματα ακρίβειας 0%. Από την άλλη πλευρά, τα μοντέλα llama3.1 ήταν σε θέση να δημιουργήσουν εκτελέσιμο κώδικα και πέτυχαν τα καλύτερα αποτελέσματα ακρίβειας σε σύγκριση με κάθε άλλο πείραμα. Επίσης, η προσθήκη των βημάτων του κώδικα που πρέπει να παράγει το μοντέλο, βελτίωσε περαιτέρω τα αποτελέσματα για αυτά τα μοντέλα. Τέλος, παρόλο που το llama3.1 πέτυχε τα καλύτερα αποτελέσματα με τη νευροσυμβολική μέθοδο, ο αυξημένος αριθμός παραμέτρων του μοντέλου llama3.1-70b δεν οδήγησε σε περαιτέρω αύξηση της ακρίβειας.

1.4 Συμπεράσματα

1.4.1 Συζήτηση

Η παρούσα διπλωματική εργασία διερεύνησε την ικανότητα των μεγάλων γλωσσικών μοντέλων (ΜΓΜ) να συμμετέχουν σε σύνθετη συλλογιστική και επίλυση προβλημάτων σε μια ποικιλία προβλημάτων παζλ και μαθηματικής συλλογιστικής. Με την κατηγοριοποίηση των παζλ σε τύπους βασισμένους σε κανόνες και χωρίς κανόνες, η εργασία αυτή παρείχε μια δομημένη προσέγγιση για την κατανόηση των διαφορετικών γνωστικών απαιτήσεων που θέτουν τα διάφοροι παζλ στα ΜΓΜ. Μέσω μιας ολοκληρωμένης επισκόπησης μεθόδων και συνόλων δεδομένων, η διατριβή αναδεικνύει επίσης το σημερινό τοπίο των επιδόσεων των LLM στην επίλυση γρίφων, μαζί με τις προκλήσεις που αντιμετωπίζουν αυτά τα μοντέλα.

Τα πειράματά μας σχεδιάστηκαν για να εμβαθύνουμε στην πρακτική εφαρμογή διαφόρων στρατηγικών και τεχνικών προτροπής σε τέσσερα σύνολα δεδομένων: GSM8K, SVAMP, RiddleSense και Game of 24. Για τα μαθηματικά σύνολα δεδομένων (GSM8K και SVAMP), οι μέθοδοι που διερευνήθηκαν περιλάμβαναν την τυπική προτροπή εισόδου-εξόδου (Ε/Ε), την ακολουθία αλυσιδωτών βημάτων σκέψης (CoT), την προτροπή από το λιγότερο στο περισσότερο σύνθετο υποπρόβλημα (LtM) και το Faithful-CoT, που αξιολογήθηκαν σε ρυθμίσεις μηδενικής μηδενικών ή μερικών παραδειγμάτων και αυτοσυνέπειας. Για τα σύνολα δεδομένων γρίφων (RiddleSense και Game of 24), συγκρίναμε την τυπική Ε/Ε, το CoT και μια προσέγγιση με βάση τη μετάφραση που μοιάζει με το Faithful-CoT, η οποία μεταφράζει τη φυσική γλώσσα σε εκτελέσιμο κώδικα.

Η βιβλιογραφική έρευνα που διεξήχθη στο πλαίσιο της παρούσας διπλωματικής παρέχει πληροφορίες σχετικά με την εφαρμογή διαφόρων τεχνικών προτροπής και την τελειοποίηση του μοντέλου σε όλους τους εξεταζόμενους τύπους παζλ. Μέθοδοι επίλυσης γρίφων όπως το ToT και το GoT ενισχύουν την πολύπλοκη συλλογιστική σε ντετερμινιστικούς γρίφους. Ωστόσο, αυτές οι τεχνικές απαιτούν μεγάλο αριθμό κλήσεων ΜΓΜ, γεγονός που μπορεί να περιορίσει την επεκτασιμότητα τους. Η έρευνα υπογραμμίζει επίσης ότι οι νευροσυμβολικές τεχνικές, όπως η μετάφραση της φυσικής γλώσσας σε κώδικα, είναι σε μεγάλο βαθμό ανεξερεύνητες σε περιβάλλοντα επίλυσης παζλ. Επιπλέον, η έλλειψη ποικιλίας σε μεθόδους και σύνολα δεδομένων για στοχαστικά παζλ με κανόνες και παζλ προγραμματισμού χωρίς κανόνες αποτελεί μια ευκαιρία για μελλοντική έρευνα.

Τα πειραματικά ευρήματα ενίσχυσαν ορισμένες από τις γνώσεις της έρευνας, αποκαλύπτοντας παράλληλα μοναδικές προκλήσεις σε όλα τα σύνολα δεδομένων. Στα σύνολα μαθηματικών δεδομένων (GSM8K και SVAMP), η απουσία προτροπών συστήματος οδήγησε σε πτώση των επιδόσεων, επιβεβαιώνοντας τη σημασία των δομημένων προτροπών για τη μαθηματική συλλογιστική. Το Faithful-CoT δεν απέδωσε βελτιώσεις στο Llama2-7b-instruct, και παρόλο που το CoT με Self-Consistency ήταν η πιο αποτελεσματική προσέγγιση, ακόμη και τα βασικά προβλήματα παρουσίασαν προκλήσεις, υποδηλώνοντας περιορισμούς στις τρέχουσες δυνατότητες συλλογισμού των μικρότερων ΜΓΜ.

Για το RiddleSense, το μέγεθος του μοντέλου αναδείχθηκε σε βασικό παράγοντα απόδοσης, με το μεγαλύτερο μοντέλο, το Llama3.1-70b, να αποδίδει καλύτερα, ενώ το CoT από μόνο του δεν εγγυάται την επιτυχία, υποδηλώνοντας την ανάγκη για πιο εξελιγμένες προσεγγίσεις πέρα από τις υπόδειξη των βημάτων του συλλογισμού. Ομοίως, στο σύνολο δεδομένων Game of 24, μόνο τα μοντέλα Llama3.1 ήταν σε θέση να δημιουργήσουν εκτελέσιμο κώδικα με τη μέθοδο Faithful-CoT, τονίζοντας τη συμβατότητα μεταξύ ορισμένων μεθόδων και της αρχιτεκτονικής των μοντέλων. Επιπλέον, το CoT δεν βελτίωσε σταθερά τα αποτελέσματα, γεγονός που υποδηλώνει ότι αυτή η προσέγγιση μπορεί να έχει περιορισμένη εφαρμογή για παζλ που απαιτούν ακριβή υπολογιστικά βήματα.

Αυτές οι πειραματικές παρατηρήσεις καταδεικνύουν ότι, ενώ τα LLM μπορούν να αντιμετωπίσουν την πολύπλοκη συλλογιστική σε έναν βαθμό, οι προκλήσεις εξακολουθούν να υφίστανται, ιδίως στην επίλυση μαθηματικών και βασισμένων σε κανόνες προβλημάτων, όπου τα μικρότερα μοντέλα συχνά δεν διαθέτουν την ικανότητα να εκτελούν με συνέπεια και αξιοπιστία σύνθετη συλλογιστική.

1.4.2 Μελλοντικές Κατευθύνσεις

Όσον αφορά τις μελλοντικές εργασίες, διάφοροι βασικοί τομείς προσφέρουν υποσχόμενες κατευθύνσεις για την προώθηση της έρευνας στην επίλυση παζλ με τη χρήση ΜΓΜ:

- **Βελτιώσεις στην Μεθοδολογία:** Ένας πρωταρχικός τομέας για μελλοντική έρευνα έγκειται στην αξιοποίηση νευροσυμβολικών τεχνικών, ιδίως της μετάφρασης φυσικής γλώσσας σε κώδικα. Αυτή η προσέγγιση, σε μεγάλο βαθμό ανεξερεύνητη σε περιβάλλοντα παζλ, θα μπορούσε να γεφυρώσει το χάσμα μεταξύ αφηρημένης συλλογιστικής και εκτελέσιμων λύσεων, ειδικά για πολύπλοκους γρίφους που απαιτούν ακριβείς λειτουργίες. Επιπλέον, υπάρχει δυνατότητα αξιολόγησης προηγμένων τοπολογιών προτροπής, όπως δομές συλλογισμού βασισμένες σε γράφους ή δέντρα, οι οποίες θα μπορούσαν να ενισχύσουν το βάθος συλλογισμού μικρότερων ΜΓΜ, διατηρώντας παράλληλα την υπολογιστική αποδοτικότητα.
- **Δημιουργία Συνόλων Δεδομένων:** Η έρευνα ανέδειξε κενά στα διαθέσιμα σύνολα δεδομένων, ιδίως για στοχαστικά παζλ και προκλήσεις προγραμματισμού. Η ανάπτυξη νέων συνόλων δεδομένων για αυτές τις κατηγορίες θα μπορούσε να τονώσει την έρευνα σε αυτούς τους τομείς, προσφέροντας ένα ευρύτερο φάσμα εργασιών για τη δοκιμή και την επέκταση των συλλογιστικών ικανοτήτων των ΜΓΜ.
- **Παραγωγή Παζλ:** Η έρευνα για την αυτόματη παραγωγή παζλ με χρήση ΜΓΜ είναι περιορισμένη, με πρόσφατες δουλειές όπως το RISCORE [92] να έχουν μόλις αρχίσει να ασχολούνται με αυτό το πεδίο. Δεδομένου ότι η κατανόηση και η επίλυση γρίφων είναι θεμελιώδους σημασίας για τη δημιουργία τους, οι εξελίξεις στις ικανότητες συλλογιστικής ΜΓΜ θα μπορούσαν να επηρεάσουν άμεσα την έρευνα για τη δημιουργία παζλ. Αυτός ο τομέας δίνει βήμα για την ανάπτυξη μοντέλων ικανών να δημιουργούν νέους γρίφους που δοκιμάζουν ποικίλες πτυχές της συλλογιστικής, της λογικής και της συμπερασματολογίας, προσθέτοντας έτσι μια νέα διάσταση στην αξιολόγηση των ΜΓΜ.

Chapter 2

Introduction

Large Language Models (LLMs) have significantly advanced natural language processing, demonstrating impressive capabilities in a wide range of tasks, including text generation, translation and question answering [84]. Models like GPT-3 [11], GPT-4 [89] and the recent Llama series [27, 117, 118] have pushed the boundaries of AI, showcasing an unique ability to understand and generate human-like text. Beyond these surface-level capabilities, a critical aspect of these models is their potential for reasoning—a cognitive process that involves making inferences, solving problems and applying logic to derive solutions [70, 69, 5, 22]. Reasoning is crucial not only for understanding context but also for tackling complex, abstract tasks that require more than rote memorization of data.

Puzzle solving serves as an ideal benchmark for evaluating the reasoning capabilities of language models. Puzzles, by design, challenge cognitive abilities such as logical reasoning, pattern recognition and strategic thinking. They require solvers to process information, make connections and apply rules creatively to arrive at solutions. This makes puzzles a rich ground for testing the depth and flexibility of an AI model's reasoning skills. Puzzles can be broadly categorized into rule-based and rule-less types, each presenting distinct challenges: rule-based puzzles, like Sudoku or Game-of-24, rely on fixed rules and structured environments, while rule-less puzzles, such as riddles and programming tasks, demand broader inferential thinking and knowledge application [38].

Despite their growing sophistication, LLMs still face notable challenges in puzzle solving. Current approaches often struggle with tasks that require multi-step reasoning, handling uncertainty, or interpreting implicit information [129, 4]. While methods like few-shot prompting [10] and Chain-of-Thought (CoT) reasoning [127, 53] have improved performance in some cases [105], significant gaps remain, particularly in puzzles that require deep logical inference or the combination of multiple reasoning steps. These limitations highlight the need for more advanced techniques that can enhance the reasoning processes of LLMs.

Investigating the reasoning capabilities of LLMs through puzzle solving provides valuable insights that can drive the development of more advanced AI systems, equipping them with the ability to handle complex reasoning tasks. Understanding how these models handle reasoning tasks can inform the development of new strategies and guide the creation of models better equipped to tackle complex real-world challenges. This thesis aims to bridge the gap between current LLM capabilities and the demands of advanced reasoning, contributing valuable insights into the development of more robust AI reasoning methods.

Our study makes two primary contributions:

1. A comprehensive survey of the existing landscape of LLM reasoning in puzzle solving, highlighting the strengths and weaknesses of various approaches.
2. Experimental evaluations of LLM performance on selected puzzles, including math datasets, RiddleSense [67] and the Game-of-24 Puzzle, using different reasoning strategies such as few-shot prompting, CoT, self-consistency and neuro-symbolic approaches.

The thesis is structured to provide a comprehensive examination of puzzle-solving using reasoning with Large

Language Models (LLMs).

In the **Background** (§3.2.3) Chapter, we describe the basic ideas of Machine Learning and Deep Learning, while **Large Language Models** (§4) Chapter analyzes the attention mechanism, the transformers architecture, the basic ideas behind LLMs, their reasoning abilities and the key methods for better prompting.

The Chapter **Puzzle Solving and LLMs** (§5) is an extensive literature review delving into the current state of research for puzzle solving using LLMs, including a categorization of puzzles and methodologies employed for solving them in previous works. [38] Key sections include:

- CATEGORIZATION OF PUZZLES: Details the types of puzzles (rule-based and rule-less) and their distinct reasoning challenges.
- METHODOLOGIES FOR PUZZLE SOLVING WITH LLMs: Discusses prompting techniques, neuro-symbolic methods and fine-tuning strategies.
- COMPARISON WITH CONVENTIONAL METHODS: Analyzes traditional approaches to puzzle solving and contrasts them with LLM-based methods.
- EVALUATION BENCHMARKS AND DATASETS FOR PUZZLES: Reviews commonly used datasets and metrics for assessing LLM performance.

The **Experiments** (§6) chapter describes the experimental setup, including the datasets used, the models and the reasoning techniques applied. Next we present the findings from the experiments, comparing the effectiveness of different approaches in solving puzzles. This chapter includes detailed analyses, visual representations and insights into the reasoning capabilities of the models.

Finally, the thesis **concludes** (§7) with the findings of this diploma thesis and highlights the potential for improved reasoning strategies and development of richer datasets for future exploration.

Chapter 3

Background

Artificial Intelligence (AI) is a broad field dedicated to creating systems capable of performing tasks that require human-like intelligence. These tasks range from recognizing patterns in visual data and interpreting spoken language to making complex decisions and understanding textual information. AI's overarching goal is to equip machines with the ability to replicate and even enhance human cognitive functions.

Within AI, Machine Learning (ML) is a critical discipline that focuses on developing algorithms and models that enable computers to learn from and make decisions based on data. Unlike traditional programming, where explicit instructions are provided for every task, ML models identify patterns and insights from large datasets, allowing them to improve their performance over time without direct human input. This capability to autonomously adapt and refine their outputs distinguishes ML from more conventional computational approaches.

As ML has advanced, it has given rise to Deep Learning (DL), a specialized subfield that leverages complex neural network architectures composed of many layers. These deep neural networks, inspired by the human brain's neural structures, are particularly adept at handling vast and complex datasets. They excel in tasks involving high-dimensional data—such as image and speech recognition, as well as natural language processing—by uncovering intricate patterns and making sophisticated decisions based on those patterns. The "depth" of these networks, or the number of processing layers, is key to their ability to manage and interpret such complex data effectively.

Contents

3.1 Machine Learning

Machine Learning (ML) is a fundamental pillar of artificial intelligence, enabling systems to learn from data, identify patterns, and make decisions with minimal human intervention. In the following subsections, we will explore the key categories of machine learning, each defined by how models learn from and interact with data.

3.1.1 Categories of Machine Learning

Machine Learning algorithms can be categorized according to the experience the model is provided with during training. The three broad types of machine learning algorithms are supervised, unsupervised and reinforcement learning, while semi- and self- supervision can be considered variants of the above.

Supervised Learning

Supervised learning is the most widely used form of machine learning, where the model is trained on a labeled dataset. Each example in the training set includes both input data in a feature vector x and the corresponding correct output y, often referred to as a "label". The goal of supervised learning is to learn a mapping from inputs to outputs that can be generalized to new, unseen data. This is achieved by minimizing the difference between the predicted output and the actual label during training and learning the distribution $p(y|x)$. Common applications of supervised learning include image classification, where the task is to label images into predefined categories, and regression tasks, such as predicting house prices based on features like size and location.

Unsupervised Learning

In contrast to supervised learning, unsupervised learning deals with data that does not have labeled outputs. Instead, they attempt to implicitly or explicitly learn the probability distribution of the entire dataset $p(x)$ and in turn give insight about the hidden structures or patterns within the data. Unsupervised learning is often used for clustering, where the aim is to group similar data points together, or for dimensionality reduction, which involves reducing the number of variables under consideration without losing important information. Algorithms like k-means clustering and principal component analysis are classic examples of unsupervised learning methods.

Semi-supervised Learning

Semi-supervised learning combines elements of both supervised and unsupervised learning. In this approach, the model is trained on a dataset that contains a small amount of labeled data and a large amount of unlabeled data. The labeled data helps guide the learning process, while the unlabeled data allows the model to capture more general patterns. Semi-supervised learning is particularly useful in situations where obtaining labeled data is expensive or time-consuming, such as in medical imaging, where only a few images may be annotated by experts, but many unlabeled images are available. This approach can significantly improve learning performance by leveraging the abundance of unlabeled data.

Self-supervised Learning

Self-supervised learning is a type of supervised learning where the system generates its own labels from the input data. This is typically done by hidding a part of the data and asking the model to predict it, effectively creating a supervised learning problem without the need for external labels. Self-supervised learning has gained prominence in training large models, particularly in natural language processing (NLP) and computer vision. For example, in natural language models, a common self-supervised task might involve predicting the next word in a sentence, which helps the model understand context and language structure. Self-supervised learning is foundational for many modern Large Language Models, such as GPT and BERT, which are pre-trained on vast amounts of text data before being fine-tuned for specific tasks.

Reinforcement Learning

Reinforcement learning (RL) is a dynamic learning approach where an agent interacts with an environment and learns to make decisions by receiving feedback in the form of rewards or penalties. The agent's goal is to maximize cumulative rewards over time by taking actions that are expected to yield the most favorable outcomes. Unlike supervised learning, where the correct output is provided for each input, reinforcement learning relies on trial and error, exploring various strategies to discover the most effective ones. RL has been successfully applied in various domains, such as robotics, where robots learn to navigate and manipulate objects, and in game playing, where agents learn to play complex games like chess and Go at a superhuman level. Although not the primary training method for LLMs, RL is sometimes used in fine-tuning stages to optimize model behaviours, such as in reinforcement learning from human feedback (RLHF), as seen in models like ChatGPT.

3.1.2 Data Modalities

In machine learning, data comes in various forms, or modalities, each with unique characteristics that influence how models are trained and applied. Understanding these different data modalities is crucial because the type of data directly impacts the choice of algorithms, the architecture of models, and the overall approach to learning. The most common data modalities include structured data, text, images, audio, video and graphs, each of which requires specific techniques and considerations.

Structured Data

Structured data is highly organized and easily interpretable by machines. It is typically stored in tabular formats such as spreadsheets or databases, where each row represents a record and each column represents a feature or attribute of the data. Structured data is often numerical or categorical, making it straightforward to process using traditional machine learning algorithms like decision trees, logistic regression, or support vector machines. Due to its organized nature, structured data allows for efficient processing and analysis.

Text Data

Text data, a form of unstructured data, presents unique challenges and opportunities in machine learning. Unlike structured data, text is inherently sequential and varies greatly in length, syntax, and vocabulary. Natural Language Processing (NLP) is the AI discipline focused on analyzing and synthesizing text data, encompassing tasks such as sentiment analysis, machine translation, and text summarization. Text data processing involves converting words into numerical representations, such as word embeddings or tokenized sequences, which models can then analyze. Techniques like bag-of-words, word2vec and transformers are commonly used to process and analyze text data. A significant advancement in processing text data has been the development of LLMs performing a wide array of language tasks with remarkable accuracy.

Image Data

Image data, another unstructured modality, has gained prominence with the advent of deep learning. Images are composed of pixels, each represented by numerical values corresponding to color intensities. Unlike text, images carry spatial information that needs to be preserved during processing. Convolutional Neural Networks (CNNs) are specifically designed to handle image data by capturing spatial hierarchies through layers of convolutional filters. These filters help in recognizing patterns such as edges, textures and more complex features, enabling tasks like object detection, image classification, and facial recognition.

Audio Data

Audio data, which includes speech and other sound signals, is inherently temporal and continuous, often represented as waveforms or spectrograms. Processing audio data requires models to capture both temporal patterns and frequency information. Speech recognition, music genre classification, and emotion detection are typical tasks that utilize audio data. Recurrent Neural Networks (RNNs), Long short-term memory models (LSTMs) and more recently transformers, are commonly used for processing audio data due to their ability to handle sequences and maintain temporal context.

Video Data

Video data combines visual and temporal elements, making it one of the most complex data modalities to process. A video can be thought of as a sequence of images (frames) accompanied by audio, where both visual and auditory components need to be analyzed. Video data is used in tasks such as action recognition, video summarization and activity detection. Processing video data typically involves combining techniques from both image and audio processing. CNNs are used to process the spatial content of each frame, while RNNs or transformers handle the temporal sequence of frames.

Graph Data

Graph data represents entities as nodes and the relationships between them as edges, capturing the intricate connections inherent in complex systems. Unlike other data modalities, graph data explicitly models relational information, making it invaluable for tasks where understanding the interconnections between entities is critical. Processing graph data typically involves specialized algorithms like Graph Neural Networks (GNNs), which extend deep learning techniques to graph-structured data, enabling tasks such as node classification, link prediction, and graph clustering.

3.2 Deep Learning

Deep Learning (DL) is a specialized subfield of machine learning that focuses on neural networks with multiple layers, allowing models to automatically learn complex representations from large datasets. Inspired by the structure and functioning of the human brain, deep learning models—often referred to as deep neural networks—consist of interconnected layers of artificial neurons that learn hierarchical patterns in data. The "depth" of a model refers to the number of layers it contains, which enables it to extract increasingly abstract features from the input data, making deep learning especially effective for tasks involving images, audio, and text.

A key characteristic of deep learning models is their ability to learn directly from raw data, reducing the need for manual feature engineering. These models excel in high-dimensional tasks, such as image recognition, speech processing, and natural language understanding, where traditional machine learning methods may struggle. For instance, the success of LLMs in NLP is largely due to the depth and complexity of their architectures, enabling them to understand and generate human-like text by capturing deep semantic and syntactic relationships.

3.2.1 Core Components

Deep neural networks are built on several fundamental components that enable them to learn and model complex relationships in data. These core elements are crucial for understanding how deep learning models function and why they are effective in diverse applications.

Neuron

At the heart of any neural network is the neuron, a computational unit that mimics the behavior of biological neurons. A neuron takes one or more inputs, processes them through a set of learned weights, and produces an output. The simplest form of a neuron is the perceptron, introduced by Frank Rosenblatt in 1958 [103]. A perceptron takes a weighted sum of its inputs, applies an activation function, and generates an output. This basic structure forms the foundation of more complex neural networks, where neurons are organized into layers and connected to each other through weighted links.

Mathematically, the output of a perceptron can be represented as:

$$y = \sigma\left(\mathbf{w}^T(\mathbf{x} + b)\right)$$

Where:

- x is the input vector,

- w is the weight vector,
- b is the bias term,
- σ is the activation function, and
- y is the output of the perceptron.

Activation Function

The activation function introduces non-linearity into the network, enabling it to learn and model complex patterns. Without non-linear activation functions, neural networks would only be able to model linear relationships between inputs and outputs, limiting their effectiveness.

Common activation functions include:

- **Sigmoid:** Maps input values to a range between 0 and 1, often used in binary classification tasks. The sigmoid function is given by the following formula:
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$
- **ReLU (Rectified Linear Unit):** ReLU outputs the input directly if positive, and zero otherwise, providing fast and effective training for deep networks. The formula is the following:
$$\text{ReLU}(x) = \max(0, x)$$
- **Tanh (Hyperbolic Tangent):** Similar to the sigmoid function but maps inputs to a range between -1 and 1, making it useful in cases where the model needs to handle negative values. The formula is the following:
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$
- **GLU (Gated Linear Unit):** GLU is an activation function designed to improve the learning of gating mechanisms in networks. It applies a gate to the input and element-wise multiplies the input and gate:
$$\text{GLU}(x) = x_1 \otimes \sigma(x_2)$$
Here, x is the input, g is the gate, and σ is the sigmoid function. GLU helps improve convergence in more complex networks by regulating information flow.
- **SiLU (Sigmoid Linear Unit):** Also known as the Swish activation function, SiLU is defined as:
$$\text{SiLU}(x) = x \cdot \sigma(x)$$
It provides smoother gradients than ReLU and has been shown to perform better in some deep networks, including LLMs.

Basic Layers

Neural networks are structured with layers that transform input data into meaningful predictions. The most common types of layers include:

- **Input Layer:** Receives the raw data for processing (e.g., pixel values for images or tokenized text for LLMs).
- **Hidden Layers:** Where most of the learning happens. These layers learn higher-level features of the data, with deeper networks extracting more abstract patterns.
- **Output Layer:** Produces the final prediction of the network. In classification tasks, it often uses a softmax function to output probabilities for each class. In LLMs, it generates the next word in a sequence for text generation tasks.

3.2.2 Training

Training a neural network involves adjusting its internal parameters (weights and biases) to minimize the difference between the predicted outputs and the actual labels of the training data. This process is iterative, with the network gradually improving its performance over time.

Loss Functions

The loss function quantifies the difference between the network's predictions and the actual target values. The goal of training is to minimize this loss, effectively improving the model's predictions. Different tasks require different loss functions, and the choice depends on whether the task is regression, classification, or more complex tasks such as sequence generation in LLMs.

- Mean Squared Error (MSE): Commonly used in regression tasks, MSE calculates the average squared difference between the predicted and actual values. The formula for MSE is:

  $$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

  Where n is the number of samples, y_i is the actual value and $\hat{y}_i$ is the predicted value.

- **Cross-Entropy Loss:** Frequently used for classification tasks, measuring the difference between two probability distributions—the true distribution and the predicted distribution. For binary classification, cross-entropy is given by:

  $$L = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right]$$

  For multi-class classification, the softmax function is typically applied in the output layer to produce a probability distribution, and the corresponding cross-entropy loss is calculated accordingly. LLMs, like GPT and BERT, often use cross-entropy loss for tasks like next-word prediction and text classification.

Backpropagation

Backpropagation is the algorithm used to compute the gradients of the loss function with respect to the model's parameters, which are then used to update the weights. This process is essential for learning in deep neural networks and relies on the chain rule to compute these gradients layer by layer, starting from the output layer and moving backward through the network.

The process consists of two main phases:

1. Forward Pass: During the forward pass, the input data is propagated through the network, and the output is generated. The loss function is then calculated based on the network's output.
2. Backward Pass: In the backward pass, the error from the loss function is propagated backward through the network. The gradient of the loss with respect to each weight is computed using the chain rule. These gradients indicate how much each weight contributed to the error.

Overfitting and Regularization

A well-trained neural network should be able to generalize to unseen data, meaning it can make accurate predictions not only on the training data but also on new, unseen examples. However, achieving good generalization requires balancing the complexity of the model and the amount of training data.

Overfitting occurs when a model performs well on the training data but fails to generalize to new data. This happens when the model becomes too complex and starts memorizing the noise in the training set rather than learning the underlying patterns. Overfitting can often be detected by a high accuracy on the training set but poor performance on the validation or test set. In the context of LLMs, overfitting can manifest as the model memorizing training sequences rather than learning general language patterns.

In order to prevent the above problem, regularization is a a set of techniques used to improve the model's ability to generalize. These methods constrain the model during training, ensuring that it doesn't become too complex and overfit the training data. Some common regularization techniques include:

- **L2 Regularization (Ridge Regression):** This technique penalizes large weights in the model by adding a term to the loss function that is proportional to the sum of the squared weights:

  $$L_{\text{total}} = L_{\text{original}} + \frac{\lambda}{2}\sum_{i=1}^{n} w_i^2$$

  where L is the original loss function and λ is a regularization parameter that controls the value of the penalty.

- **Dropout:** During training, dropout randomly sets to zero a fraction of the neurons in the network, forcing the model to learn more robust features. This prevents the network from relying too much on any particular neuron and helps reduce overfitting.

- **Early Stopping:** Early stopping involves halting the training process when the performance on the validation set starts to deteriorate, even if the training loss continues to decrease.

Pre-training and Fine-tuning

Modern deep learning models, particularly LLMs, often go through two stages of training:

- **Pre-training:** During pre-training, the model is trained on a large, general dataset in a self-supervised manner, learning general features of the data. In the case of LLMs, this involves training on massive corpora of text to learn word embeddings and the general structure of language. For example, GPT models are pre-trained to predict the next word in a sentence, learning linguistic patterns without the need for manually labeled data.

- **Fine-tuning:** After pre-training, the model is fine-tuned on a smaller, task-specific dataset. Fine-tuning adapts the model's knowledge to a particular task, such as sentiment analysis, question answering, or puzzle solving. Fine-tuning allows the model to perform well on specific tasks while leveraging the broad knowledge acquired during pre-training.

3.2.3 Evaluation

Evaluating a neural network is a significant part, ensuring that the model generalizes well to unseen data and performs effectively in real data. This process typically involves dividing the data into separate sets and using performance metrics to measure how well the model achieves its objectives.

Training, Validation, and Test Sets

When training a neural network, the dataset is typically split into three subsets:

- **Training Set:** During training, the model adjusts its internal parameters (weights and biases) to minimize the error on the training data. The model "learns" from this data by iteratively adjusting its parameters to reduce the loss function.

- **Validation Set:** The validation set is used during training to monitor the model's performance on data it has not yet seen. This helps in tuning hyperparameters, such as the learning rate and the number of layers or neurons in the network. The model is not directly trained on the validation data, so its performance on this set provides an indication of how well it generalizes to new data and avoids overfitting.

- **Test Set:** After training is complete, the model is evaluated on the test set, which represents the final unseen data. This set is used to assess the model's true generalization capabilities, as it contains data that was never used during training or validation.

Performance Metrics

The most widely used metrics to evaluate model's predictions include accuracy, precision, recall, and F1 score, each providing a different perspective on model performance.

- **Accuracy** measures the proportion of correct predictions out of the total number of predictions. It is a simple and intuitive metric but may not be suitable for imbalanced datasets where one class dominates the others.
- **Precision** measures the proportion of true positive predictions out of all positive predictions made by the model. It is particularly useful in scenarios where false positives are costly, such as in medical diagnosis.
- **Recall** measures the proportion of actual positives that were correctly identified by the model. High recall is crucial when the cost of missing a positive instance (false negative) is high, such as in disease detection.
- The **F1 score** is the harmonic mean of precision and recall, providing a balanced measure when both precision and recall are important. It is especially useful when the data is imbalanced.

These performance metrics are critical for evaluating the generalization capability of the model. In LLMs, which often perform tasks like text classification or question answering, metrics like accuracy and F1 score are commonly used to assess the quality of predictions. For example, in a text classification task, accuracy might reflect how often the model classifies a document correctly, while the F1 score might be used to balance precision and recall, particularly when one class is much more frequent than others. For most of the puzzle benchmarks, the applied metric is the success rate of the model for solving the puzzle.

Chapter 4

Large Language Models (LLMs)

Large Language Models (LLMs) are a subset of deep learning models that have reshaped the landscape of artificial intelligence, particularly in natural language processing (NLP), due to their ability to generate, understand, and interact with human language at an unprecedented scale. These large, general-purpose models are pre-trained on massive datasets containing a wide array of textual information from books, articles, and internet sources, enabling them to learn intricate linguistic patterns and structures. Following pre-training, LLMs are often fine-tuned for specific tasks or domains, allowing them to adapt their general language understanding to specialized applications with minimal additional training data.

The rise of LLMs represents a significant leap from earlier neural models, introducing capabilities that extend beyond simple text generation to encompass complex tasks such as reasoning, problem-solving, and creative writing. LLMs can handle a wide range of language-related problems, including text classification, question answering, document summarization, language translation, sentiment analysis, and natural conversation. Their versatility makes them valuable across various industries, where they provide customized AI solutions that offer advantages in speed, accuracy, flexibility, and ease of deployment compared to older AI technologies.

The foundation of LLMs lies in their ability to model language through deep learning architectures, with the most influential being the transformer model introduced by Vaswani et al. (2017) [119]. Transformers utilize self-attention mechanisms, allowing models to efficiently process and generate sequences of text by capturing dependencies between words regardless of their position in the sequence. This architecture is pivotal in enabling LLMs to recognize relationships and connections within text, enhancing their understanding and generation capabilities. The training process of LLMs typically begins with unsupervised learning, where the model learns to establish foundational relationships between words and concepts by predicting the next word in a sequence based on prior context. This initial training phase is often followed by supervised fine-tuning, which refines the model's outputs for specific tasks using labeled data.

Unlike previous architectures such as Recurrent Neural Networks (RNNs) [108] and Long Short-Term Memory (LSTM) networks [43, 108], transformers do not require sequential data processing, making them highly scalable and suitable for training on large datasets. This shift to transformer-based architectures has been a key enabler of the success of modern LLMs, as they are capable of learning nuanced relationships and contextual information that drive their performance across a wide range of applications. The vast parameter counts and large-scale datasets used in training allow these models to handle complex language tasks with impressive generalization abilities.

Once trained, LLMs serve as the foundation for a wide spectrum of AI applications, demonstrating capabilities that often surpass traditional NLP benchmarks. Compared to earlier AI chatbot technologies, LLMs provide superior performance in generating coherent and contextually appropriate responses, demonstrating advanced reasoning and problem-solving skills. This has positioned LLMs not only as powerful tools for language understanding but also as emerging contenders in cognitive tasks traditionally reserved for human intelligence. However, despite their remarkable performance in many domains, LLMs still face significant challenges when it comes to tasks that require deep logical reasoning, multi-step problem-solving, and the interpretation of implicit knowledge. Thus, the field of explainable AI (XAI) plays a key role in the safety and usefulness of such systems [77, 141, 31, 56]

The exploration of reasoning capabilities in these models is particularly relevant as it touches on their potential to simulate human-like thought processes, a crucial aspect of artificial general intelligence (AGI). This chapter aims to provide a comprehensive overview of LLMs, focusing on their development, architecture, and capabilities in reasoning. We will delve into the categories of LLMs, explore the underlying principles of transformers, and examine how these models handle complex reasoning tasks. The following sections outline the key topics covered:

Contents

4.1 Embeddings

Embeddings are a foundational element in LLMs, representing a method of converting words or phrases into dense, continuous vectors. These vectors capture the semantic meaning and relationships between words, enabling models to process text in a way that reflects the nuances of natural language. Instead of representing words as one-hot encoded vectors—which fail to capture similarities between words—embeddings encode words into fixed-size vectors that reflect their meaning in the context of a given dataset.

Word Embeddings

Early models, such as Word2Vec [83] and GloVe [95], were among the first to introduce word embeddings. These methods train embeddings based on co-occurrence statistics, learning to represent words in a continuous vector space where semantically similar words are positioned close to each other. For example, the words "brother" and "sister" would be close in the vector space, as would "uncle" and "aunt", as shown in Figure 4.1.1.

The main advantage of word embeddings lies in their ability to capture semantic relationships, such as synonyms and analogies, in a way that traditional one-hot encoding cannot. However, these early embeddings were static, meaning each word had a single fixed vector representation, regardless of its context in a sentence.

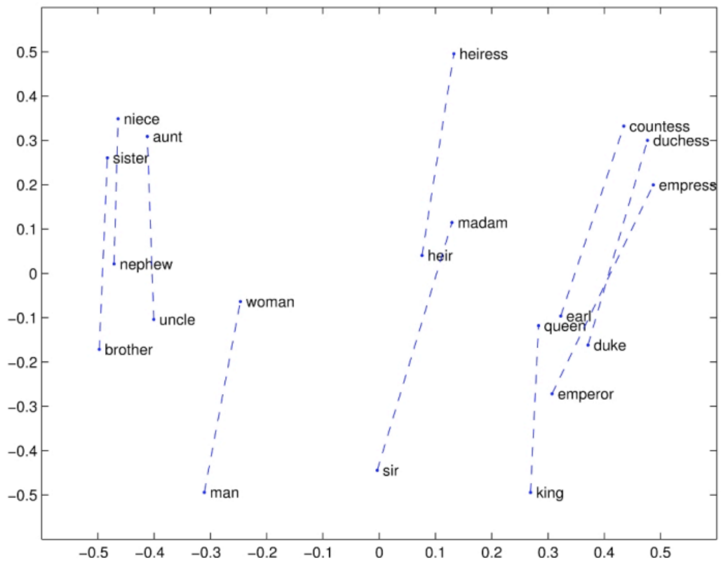


Figure 4.1.1: Glove Visualizations by Richard Socher [95]

Contextual Embeddings

In modern LLMs, embeddings have evolved to become contextual. Models like ELMo [96], BERT [24] and GPT [99] generate embeddings based on the context in which a word appears. This means that the word "bank" would have different vector representations in the sentences "I went to the bank to withdraw money" and "The river overflowed its bank."

Contextual embeddings allow models to better capture the meaning of words in different contexts, vastly improving performance on complex NLP tasks such as question answering, text generation, and translation. In LLMs, contextual embeddings are generated during the pre-training phase, where models are exposed to large amounts of text and learn to predict words based on their surrounding context.

Tokenization

Before embeddings can be generated, the raw text data must be tokenized, which is the process of splitting text into smaller units called tokens. These tokens serve as the input to LLMs and can represent words, subwords, or even individual characters, depending on the tokenization strategy. The tokenization process is essential because it allows the model to handle language in a structured way, transforming text into numerical representations that the model can process.

There are various tokenization strategies:

- **Word-level Tokenization:** In word-level tokenization, each word in the text is treated as a single token. For example, the sentence "Solving puzzles requires logical thinking" would be tokenized as ["Solving", "puzzles", "requires", "logical", "thinking"]. While this approach is simple it requires a large vocabulary to cover all possible words, and any words not present in the vocabulary are treated as unknown, often represented as <UNK>.
- **Subword Tokenization:** Modern LLMs, employ subword tokenization methods like Byte Pair Encoding (BPE) [34] and WordPiece [24]. In these approaches, words are broken down into smaller units, such as prefixes, suffixes, or other meaningful subwords. This allows the model to efficiently handle rare words and morphological variations by decomposing them into more common subwords. For example, the word "puzzles" might be tokenized as ["puzz", "##les"], where "##les" is a subword that can be combined with different roots (e.g., "tab", "sub") to form other words like "tables" or "subtitles."
- **Character-level Tokenization:** In character-level tokenization, each character in the input text is treated as a token. For example, the word "puzzle" would be tokenized as ["p", "u", "z", "z", "l", "e"]. While this approach avoids the need for large vocabularies and can theoretically handle any word or sentence, it significantly increases the length of the input sequence.
- **SentencePiece:** SentencePiece is another tokenization method used by models such as Google's T5 [102]. It is designed to tokenize text without requiring explicit word boundaries, which makes it language-agnostic and more robust for handling languages that lack clear word delimiters (such as Chinese or Japanese).

In summary, tokenization and embeddings play a crucial role in the performance of LLMs. By breaking text into manageable pieces, tokenization transforms raw input into sequences that can be mapped to embeddings, providing the model with a dense representation of words and their meanings.

4.2 Transformers

Transformers are the foundational architecture behind modern LLMs, such as GPT, Llama, Mistral, BERT, and T5. Introduced by Vaswani et al. (2017) [119], transformers leverage the attention mechanism to model relationships between different parts of a sequence, enabling them to process text more efficiently than previous architectures like RNNs or LSTMs. In this section, we explore the core components of transformers, including the attention mechanism, self-attention, and the pretraining architectures that have made LLMs so effective in a wide variety of tasks.

4.2.1 Attention

The attention mechanism was originally developed to improve the performance of neural networks on sequence-based tasks by allowing the model to focus on the most relevant parts of the input. Attention helps models capture dependencies between distant elements in a sequence, enabling them to weigh the importance of different input tokens based on their relevance to a given task.

In its simplest form, attention can be understood as a weighted sum of input features, where the weights represent how much attention the model should pay to each part of the sequence. Formally, the attention function can be described as:

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

Where:

- Q (Query), K (Key), and V (Value) are the input vectors,
- d_k is the dimensionality of the key vectors,
- QK^T calculates the dot product of the query and key vectors to determine the similarity between tokens, and
- The softmax function converts these similarities into attention weights.

Attention mechanisms, such as *global attention*, allow the model to consider every word in the sequence when predicting the next word, while *local attention* restricts the model to a limited window of tokens.

The main advantage of attention over previous methods like RNNs or LSTMs is that it allows the model to capture long-range dependencies between tokens without the need for sequential processing. This makes attention highly efficient, as it can process all tokens in parallel.

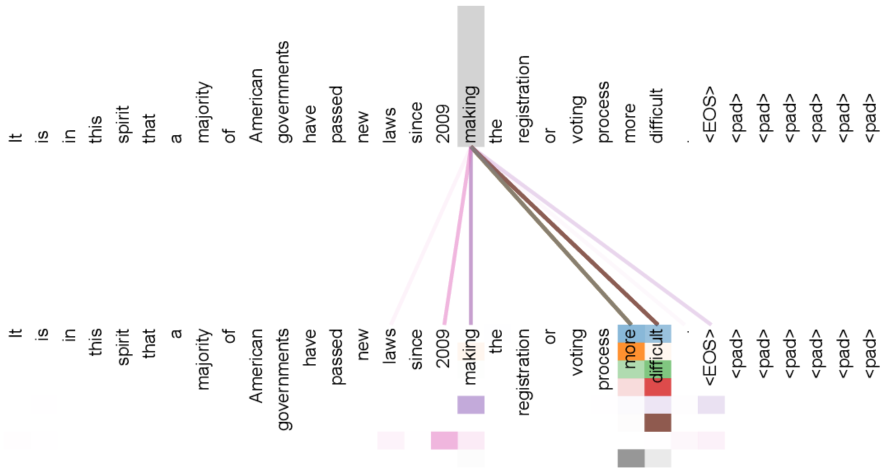


Figure 4.2.1: An example of the attention mechanism [119]

4.2.2 Self-Attention and Transformer Networks

Self-attention extends the attention mechanism by allowing each token in the input sequence to attend to every other token, including itself. This approach enables models to capture relationships between all tokens in a sequence, regardless of their positions. In contrast to traditional attention mechanisms, which require external input (queries) to generate attention scores, self-attention uses the same input sequence for queries, keys, and values.

The process of self-attention can be broken down into three steps:

1. **Calculating Query, Key, and Value vectors:** For each token in the input sequence, self-attention generates three vectors: a query, a key, and a value.
2. **Computing Attention Scores:** The attention scores for each token are computed by taking the dot product of its query with all keys, followed by a softmax operation to generate attention weights.
3. **Aggregating Values:** Each token's final representation is computed as a weighted sum of the value vectors, where the weights are determined by the attention scores.

In transformer networks, this self-attention mechanism is applied at each layer of the model, allowing the network to build increasingly abstract representations of the input data. These representations capture not only the meaning of individual words but also their relationships to other words in the sequence.

Transformer models consist of multiple encoder and decoder layers:

- **Encoder Layers:** Each encoder layer includes a multi-head self-attention mechanism followed by a feed-forward neural network. The encoder is responsible for generating rich representations of the input.
- **Decoder Layers:** The decoder layers also include multi-head self-attention, but they additionally use *cross-attention* to focus on relevant parts of the encoder's output. This is crucial for tasks like machine translation, where the model needs to attend to both the input and the output during generation.

Multi-Head Self-Attention

One of the key innovations of the transformer model is multi-head self-attention, which allows the model to compute attention multiple times in parallel, each time with different sets of query, key, and value vectors. This enables the model to capture different types of relationships in the data. The outputs from each head are concatenated and passed through a feed-forward layer to produce the final output.

Self-attention and transformers have revolutionized natural language processing, enabling models to process large-scale datasets efficiently and learn contextual representations that are crucial for tasks like text generation, summarization, translation, and even reasoning.

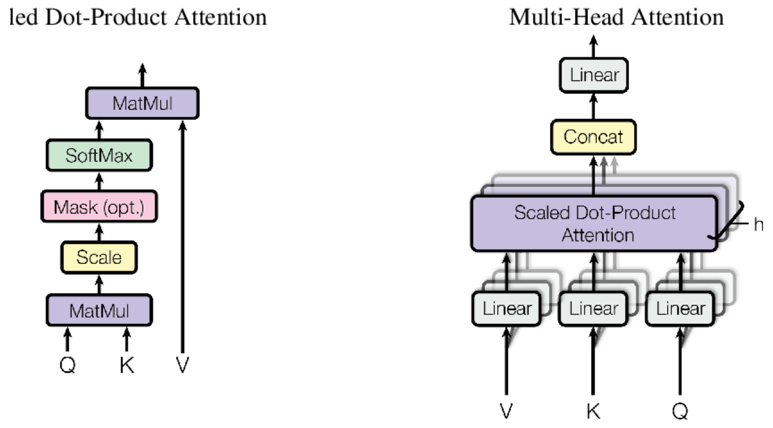


Figure 4.2.2: Scaled Dot-Product and Multi-Head Attention [119]

4.2.3 Pretraining Architectures

As we discussed earlier, pretraining on lanugage model tasks is a form of self-supervised learning. Self-supervised learning takes raw (unlabeled) data, removes information and trains a model to recover that information. In the case of language modeling, the information removed is the next word, so the model is trained to predict future words given the context. Pretraining enables models to learn general language features before being fine-tuned for specific tasks. The most widely used pretraining architectures include autoregressive (pretrained decoders), autoencoding (pretrained encoders), and sequence-to-sequence (pretrained encoder-decoders) models.

Pretrained Decoders (Autoregressive Models)

In autoregressive models like the GPT series [99, 100, 10, 89], the model is trained to predict the next word in a sequence based solely on the previous words. This unidirectional approach allows the model to generate coherent text by sampling one token at a time. Autoregressive models function as pretrained decoders in a transformer architecture, where the decoder learns to predict subsequent tokens in a left-to-right manner.

Autoregressive models excel in text generation tasks, such as dialogue systems and story generation, because of their ability to generate sequential text with coherent context.

Due to their unidirectional nature, they do not have access to the full context of the sequence (i.e. future tokens) during training, which can limit their understanding of some tasks that require full context comprehension.

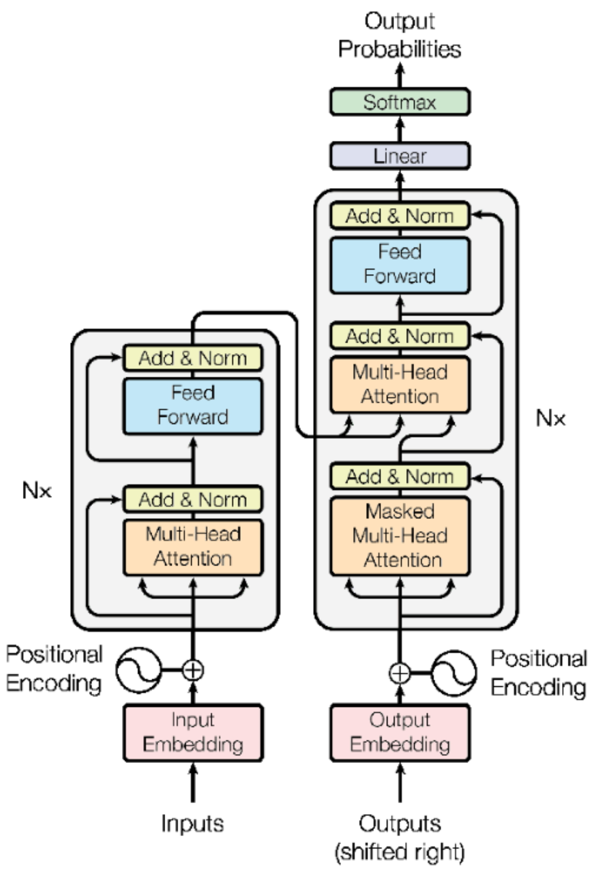


Figure 4.2.3: Transformer Architecture [119]

The GPT (Generative Pretrained Transformer) and Llama series some of the most prominent examples of pretrained decoders, where the model is pretrained on large corpora using an autoregressive language modeling objective—predicting the next word based on the previous ones. In this thesis, we will primarily focus on this category of models for the experiments and evaluations discussed in later chapters, particularly for tasks involving reasoning, lateral thinking and puzzle solving.

Pretrained Encoders (Autoencoding Models)

In autoencoding models like BERT [24], the model is trained to reconstruct missing or masked words in a sentence by attending to the context from both directions. These models operate as pretrained encoders, where the primary task is to encode the input text into rich, bidirectional representations by leveraging self-attention across all tokens in the input.

Autoencoding models are ideal for tasks requiring a deep understanding of the input text, such as text classification, sentiment analysis, and question answering. They are particularly strong in tasks where the full context (i.e. both the left and right sides of a word) is necessary for understanding the input.

While they excel in understanding, these models are not designed for generating coherent text, as they focus primarily on encoding rather than decoding.

BERT (Bidirectional Encoder Representations from Transformers) is a classic pretrained encoder model. During pretraining, BERT uses a masked language modeling (MLM) objective, where random words in a sentence are masked, and the model is trained to predict the masked words based on surrounding context.

Pretrained Encoder-Decoders (Sequence-to-Sequence Models)

Sequence-to-sequence models like T5 [102] and BART [65] leverage both encoder and decoder networks to handle tasks that require mapping an input sequence to an output sequence. In these models, the encoder processes the input sequence and generates an intermediate representation, which the decoder then uses to generate the output sequence. These models operate as pretrained encoder-decoders, allowing for bidirectional encoding of the input and unidirectional decoding of the output.

Pretrained encoder-decoder models are very flexible and excel in tasks like translation, summarization, and text-to-text transformations. They are well-suited for tasks that require generating an output sequence from an input sequence, such as converting a question to an answer, summarizing a long document, or translating text between languages.

Due to their complexity (involving both encoding and decoding), they require more computational resources compared to pretrained encoders or decoders alone.

T5 (Text-to-Text Transfer Transformer) is a versatile pretrained encoder-decoder model, where all NLP tasks are framed as text-to-text problems. For instance, the task of translation would involve taking an English sentence as input and generating a French sentence as output. BART (Bidirectional and Auto-Regressive Transformers) is another example of a pretrained encoder-decoder that uses bidirectional encoding and autoregressive decoding, making it effective for text generation and summarization.

4.3 Prompting

A *prompt* functions as the input for a generative AI system, outlining the specific task or objective that the system is expected to execute. The prompt guides the model's behavior by framing the context, ensuring that the AI understands what is required of it. *Prompting* refers to the process by which users interact with Large Language Models, providing them with specific instructions or input formats to elicit desired outputs. This approach offers a significant advantage over traditional methods, such as fine-tuning or retraining, as it allows users to customize model behavior without modifying the model's underlying parameters. Instead of the time-consuming process of retraining a model on task-specific data, a well-crafted prompt can guide the model toward the correct outcome using its pre-existing knowledge.

One of the primary advantages of prompting is its efficiency. Since no additional training or fine-tuning is required, prompting can yield immediate results with minimal computational resources. This makes it highly effective for cases where users need to adapt models to new tasks quickly. Moreover, prompting is flexible and task-agnostic; with the right input, LLMs can be directed to perform a wide variety of tasks, including text generation, reasoning, question answering, and problem-solving, without requiring task-specific models. This flexibility contrasts with traditional approaches where separate models might be needed for different tasks, each requiring specialized training.

However, prompting also comes with certain limitations. While it is a powerful way to harness the capabilities of LLMs, it relies heavily on the model's pre-existing knowledge, which can be incomplete or biased. Additionally, crafting an effective prompt can be challenging, as slight changes in wording or format can dramatically impact the model's performance. In some cases, the model may misinterpret vague or poorly-structured prompts, leading to suboptimal outputs. This sensitivity requires expertise in prompt engineering, the process of designing and refining prompts to achieve the best possible results. In contrast, fine-tuned models are more directly optimized for specific tasks and can sometimes provide more accurate and reliable results.

The success of prompting depends on several factors, such as how well the prompt aligns with the model's training, the clarity of the instructions, and whether relevant examples are provided. Despite its limitations, prompting remains a highly efficient and scalable method for leveraging the capabilities of LLMs. This section explores some basic prompt categories and techniques used to optimize LLM performance in general, involving several tasks in NLP and multimodal learning [115, 58, 57, 93, 3]. This section explores various prompt categories and techniques used to optimize LLM performance, while a more in-depth analysis of how prompting is applied specifically for reasoning and puzzle-solving tasks will be presented in a later chapter.

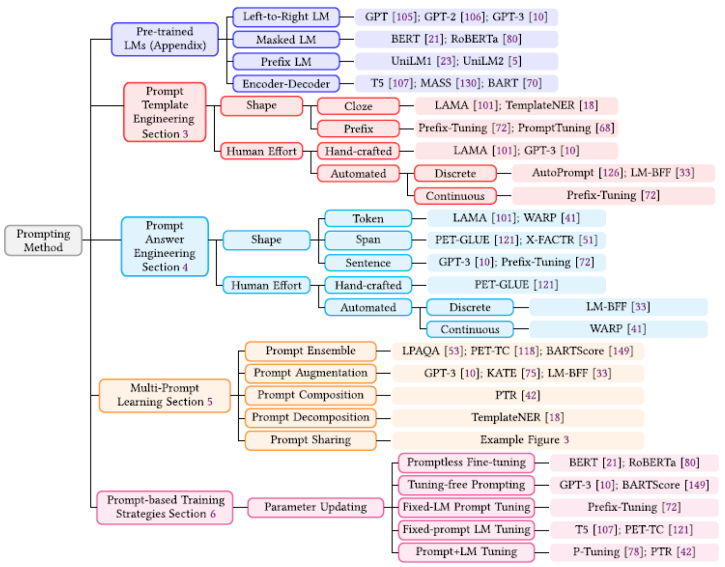


Figure 4.3.1: Prompting Typology [71]

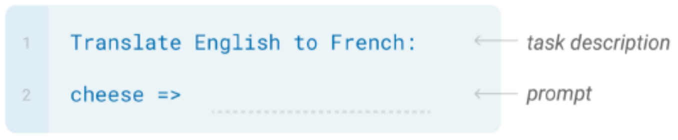
4.3.1 Prompt Categories

Liu et al. (2021) [71] categorize prompting methods as shown in the Figure 4.3.1. However, for now we will follow a categorization based on their structure and intent, with different types of prompts serving different purposes depending on the task. The following categories provide an overview of how prompting can be applied to guide LLMs:

Zero-Shot Prompting

In zero-shot prompting, the model is asked to perform a task without any specific examples being provided. This approach tests the model's ability to generalize based on its pre-trained knowledge. Zero-shot prompting is useful for tasks where users want to gauge the model's general knowledge or its ability to handle new tasks without prior examples. For instance, in puzzle-solving, the model might be asked to solve a riddle or logic problem without any prior context or examples, like the following:

Solve the puzzle: *What has many keys but can't open locks?*



```
Translate English to French:
cheese =>
```

Figure 4.3.2: Zero-shot prompting [10]

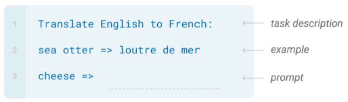
Few-Shot Prompting

Few-shot prompting [10] enhances the model's ability to perform a task by providing it with a few examples of how the task should be done. When only one example is provided then the prompting is called one-shot. The model uses these examples as a guide for generating responses to new inputs. In puzzle-solving, this might involve showing the model how to solve a few similar puzzles before asking it to solve a new one, like the following examples:

Example 1:
Puzzle: What has keys but can't open locks?
Answer: A piano.

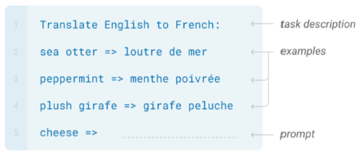
Example 2:
Puzzle: What can travel around the world while staying in the same corner?
Answer: A stamp.

Now solve the following:
Puzzle: What has many needles but doesn't sew?
Answer:



```
Translate English to French:
sea otter => loutre de mer
cheese =>
```

(a) One-shot prompting [10]



```
Translate English to French:
sea otter => loutre de mer
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>
```

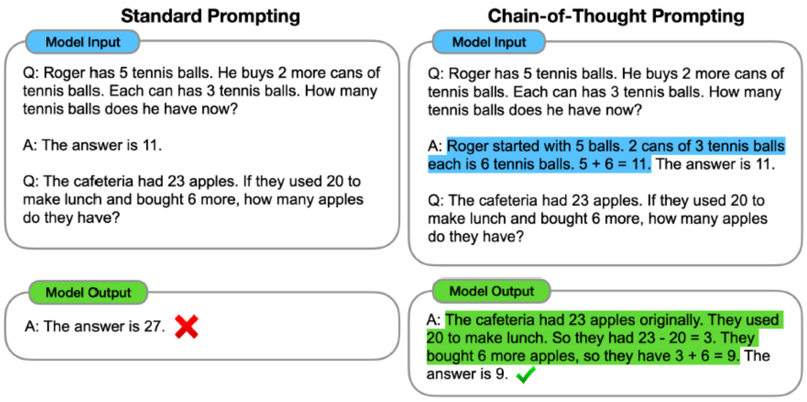
(b) Few-shot prompting [10]

4.3.2 Prompting Techniques

Several techniques have been developed to optimize prompting for specific tasks. These techniques guide the model in performing more complex operations, including reasoning, problem-solving, and step-by-step thinking. Below are some commonly used techniques:

Chain-of-Thought (CoT)

Chain-of-Thought (CoT) [53, 127] prompting is a technique that encourages the model to break down its reasoning process into individual steps before arriving at a final answer. This approach is particularly effective in puzzle-solving tasks, where reasoning is crucial. By guiding the model to outline each step, CoT improves both the accuracy and clarity of the model's response.



Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

A: The answer is 27. ✗

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

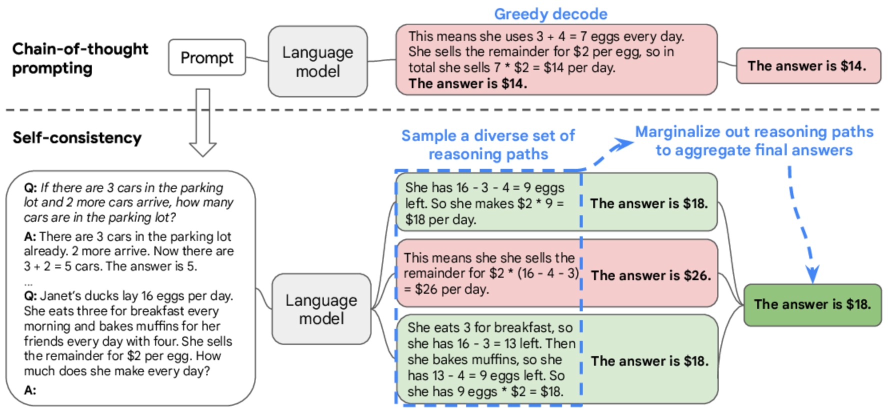
Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓

Figure 4.3.4: Chain-of-thought [127]

Self-Consistency CoT

Self-Consistency CoT (SC-CoT) [124] is a variation of Chain-of-Thought prompting that generates multiple reasoning paths and selects the most consistent outcome. This technique leverages the model's ability to propose multiple solutions, compare them, and then choose the one that appears most reliable based on patterns of consistency across reasoning chains. This method has been shown to improve the reliability and accuracy of LLM outputs, particularly in tasks requiring complex reasoning, such as puzzle-solving and multi-step logic tasks.



This means she uses 3 + 4 = 7 eggs every day. She sells the remainder for $2 per egg, so in total she sells 7 * $2 = $14 per day. **The answer is $14.**

Q: *If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot?*
A: There are 3 cars in the parking lot already. 2 more arrive. Now there are 3 + 2 = 5 cars. The answer is 5.
...
Q: Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder for $2 per egg. How much does she make every day?
A:

She has 16 - 3 - 4 = 9 eggs left. So she makes $2 * 9 = $18 per day.

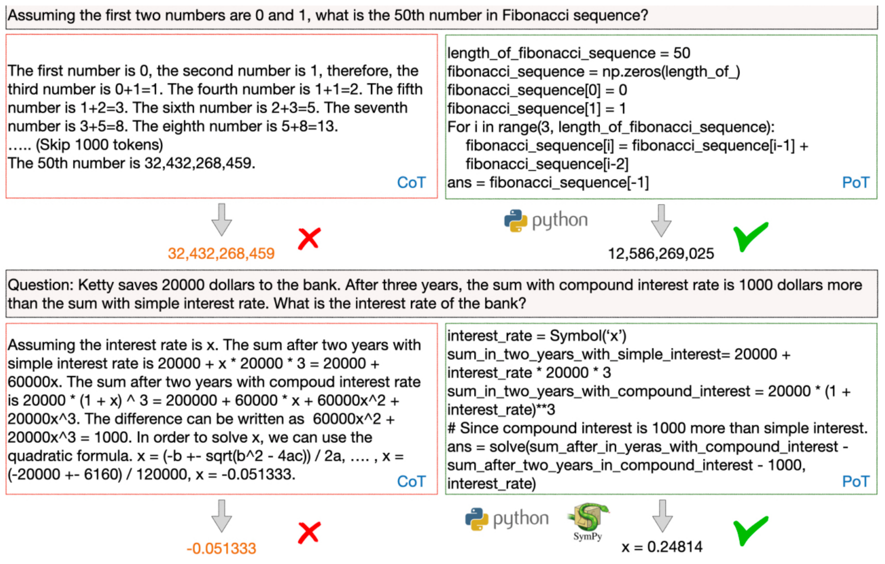
This means she she sells the remainder for $2 * (16 - 4 - 3) = $26 per day.

She eats 3 for breakfast, so she has 16 - 3 = 13 left. Then she bakes muffins, so she has 13 - 4 = 9 eggs left. So she has 9 eggs * $2 = $18.

Figure 4.3.5: Self-Consistency CoT [124]

Program-of-Thoughts (PoT)

Program-of-Thoughts (PoT) [16] prompting involves guiding the model to approach a problem in a systematic, rule-based way, similar to writing a program or algorithm. This method is particularly effective in puzzle-solving tasks that involve logical sequences or structured reasoning.



Assuming the first two numbers are 0 and 1, what is the 50th number in Fibonacci sequence?

The first number is 0, the second number is 1, therefore, the third number is 0+1=1. The fourth number is 1+1=2. The fifth number is 1+2=3. The sixth number is 2+3=5. The seventh number is 3+5=8. The eighth number is 5+8=13.
..... (Skip 1000 tokens)
The 50th number is 32,432,268,459.

```
length_of_fibonacci_sequence = 50
fibonacci_sequence = np.zeros(length_of_)
fibonacci_sequence[0] = 0
fibonacci_sequence[1] = 1
For i in range(3, length_of_fibonacci_sequence):
    fibonacci_sequence[i] = fibonacci_sequence[i-1] +
    fibonacci_sequence[i-2]
ans = fibonacci_sequence[-1]
```

Question: Ketty saves 20000 dollars to the bank. After three years, the sum with compound interest rate is 1000 dollars more than the sum with simple interest rate. What is the interest rate of the bank?

Assuming the interest rate is x. The sum after two years with simple interest rate is 20000 + x * 20000 * 3 = 20000 + 60000x. The sum after two years with compoud interest rate is 20000 * (1 + x) ^ 3 = 200000 + 60000 * x + 60000x^2 + 20000x^3. The difference can be written as 60000x^2 + 20000x^3 = 1000. In order to solve x, we can use the quadratic formula. x = (-b +- sqrt(b^2 - 4ac)) / 2a, ... , x = (-20000 +- 6160) / 120000, x = -0.051333.

```
interest_rate = Symbol('x')
sum_in_two_years_with_simple_interest= 20000 +
interest_rate * 20000 * 3
sum_in_two_years_with_compound_interest = 20000 * (1 +
interest_rate)**3
# Since compound interest is 1000 more than simple interest.
ans = solve(sum_after_in_yeras_with_compound_interest -
sum_after_two_years_in_compound_interest - 1000,
interest_rate)
```

Figure 4.3.6: Program-of-thoughts [16]

Prompt Tuning

Prompt Tuning [63] is an advanced technique that allows models to adapt to specific tasks by learning "soft prompts." Unlike traditional discrete prompts where users input natural language text (as in GPT-3), soft prompts are learned through backpropagation and fine-tuned on a small task-specific dataset. This allows the model to adapt more precisely to a task without updating the model's internal parameters, maintaining efficiency. The main benefit of prompt tuning is that it stores small task-specific prompts for each task, enabling mixed-task inference using the same pre-trained model.

In the context of puzzle-solving, prompt tuning can be especially useful in adapting LLMs to domain-specific reasoning challenges. For example, if the task involves solving complex logical or mathematical puzzles, the model can learn specific patterns or clues from a limited number of labeled examples and apply them across various puzzle types without needing full fine-tuning.

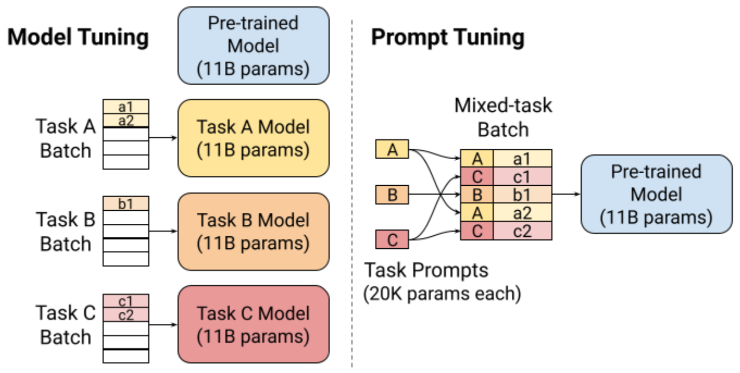

Model Tuning
Pre-trained Model (11B params)
Task A Batch
a1
a2
Task A Model (11B params)
Task B Batch
b1
Task B Model (11B params)
Task C Batch
c1
c2
Task C Model (11B params)
Prompt Tuning
Mixed-task Batch
A a1
C c1
B b1
A a2
C c2
A
B
C
Pre-trained Model (11B params)
Task Prompts (20K params each)

Figure 4.3.7: Model Tuning vs Prompt Tuning [63]

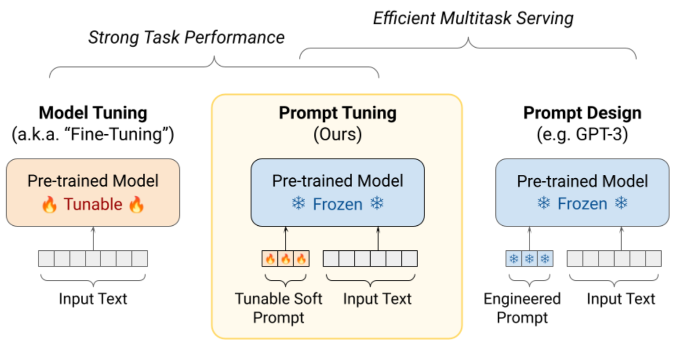


Figure 4.3.8: Prompt Tuning [63]

4.4 LLMs and Reasoning

Large Language Models have shown remarkable capabilities in understanding and generating text across a wide range of tasks. However, reasoning, which requires applying logic, inference, and step-by-step thinking, remains a significant challenge for these models. The field of reasoning with LLMs explores how these models can be optimized to solve tasks that involve complex cognitive processes, such as puzzles, multi-step logic problems, and abstract reasoning. In this section, we will explore the state of reasoning in LLMs, current limitations, and various strategies to enhance their reasoning capabilities.

While prompting techniques, such as Chain-of-Thought, can enhance a model's performance on reasoning tasks, as discussed in the previous section, LLMs still struggle with multi-step reasoning and logical inference, especially when presented with out-of-distribution (OOD) tasks. Recent research [130, 4] suggests that more robust reasoning frameworks are needed to improve LLM performance on such tasks.

4.4.1 Categories of Reasoning with LLMs

Based on the work of Qiao et al. (2022) [98] reasoning methods are divided into two main categories: Strategy Enhanced Reasoning and Knowledge Enhanced Reasoning. These methods represent different approaches to prompting LLMs to improve their reasoning performance.

Strategy Enhanced Reasoning

Strategy enhanced reasoning focuses on the process by which LLMs can be guided to perform logical reasoning more effectively. These methods use external engines or systematic frameworks to guide the reasoning process. Besides prompt engineering, which been discussed earlier, some key strategies include:

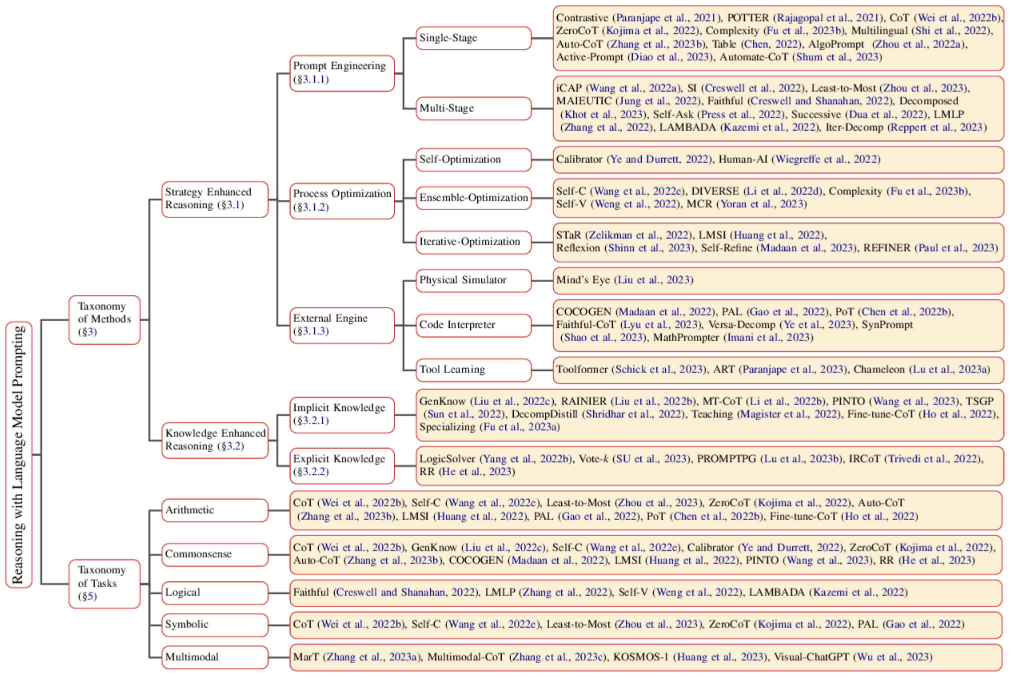

Reasoning with Language Model Prompting
Taxonomy of Methods (§3)
Strategy Enhanced Reasoning (§3.1)
Prompt Engineering (§3.1.1)
Single-Stage
Contrastive (Paranjape et al., 2021), POTTER (Rajagopal et al., 2021), CoT (Wei et al., 2022b), ZeroCoT (Kojima et al., 2022), Complexity (Fu et al., 2023b), Multilingual (Shi et al., 2022), Auto-CoT (Zhang et al., 2023b), Table (Chen, 2022), AlgoPrompt (Zhou et al., 2022a), Active-Prompt (Diao et al., 2023), Automate-CoT (Shum et al., 2023)
Multi-Stage
iCAP (Wang et al., 2022a), SI (Creswell et al., 2022), Least-to-Most (Zhou et al., 2023), MAIEUTIC (Jung et al., 2022), Faithful (Creswell and Shanahan, 2022), Decomposed (Khot et al., 2023), Self-Ask (Press et al., 2022), Successive (Dua et al., 2022), LMLP (Zhang et al., 2022), LAMBADA (Kazemi et al., 2022), Iter-Decomp (Reppert et al., 2023)
Process Optimization (§3.1.2)
Self-Optimization
Calibrator (Ye and Durrett, 2022), Human-AI (Wiegreffe et al., 2022)
Ensemble-Optimization
Self-C (Wang et al., 2022e), DIVERSE (Li et al., 2022d), Complexity (Fu et al., 2023b), Self-V (Weng et al., 2022), MCR (Yoran et al., 2023)
Iterative-Optimization
STaR (Zelikman et al., 2022), LMSI (Huang et al., 2022), Reflexion (Shinn et al., 2023), Self-Refine (Madaan et al., 2023), REFINER (Paul et al., 2023)
External Engine (§3.1.3)
Physical Simulator
Mind's Eye (Liu et al., 2023)
Code Interpreter
COCOGEN (Madaan et al., 2022), PAL (Gao et al., 2022), PoT (Chen et al., 2022b), Faithful-CoT (Lyu et al., 2023), Versa-Decomp (Ye et al., 2023), SynPrompt (Shao et al., 2023), MathPrompter (Imani et al., 2023)
Tool Learning
Toolformer (Schick et al., 2023), ART (Paranjape et al., 2023), Chameleon (Lu et al., 2023a)
Knowledge Enhanced Reasoning (§3.2)
Implicit Knowledge (§3.2.1)
GenKnow (Liu et al., 2022c), RAINIER (Liu et al., 2022b), MT-CoT (Li et al., 2022b), PINTO (Wang et al., 2023), TSGP (Sun et al., 2022), DecompDistill (Shridhar et al., 2022), Teaching (Magister et al., 2022), Fine-tune-CoT (Ho et al., 2022), Specializing (Fu et al., 2023a)
Explicit Knowledge (§3.2.2)
LogicSolver (Yang et al., 2022b), Vote-k (SU et al., 2023), PROMPTPG (Lu et al., 2023b), IRCoT (Trivedi et al., 2022), RR (He et al., 2023)
Taxonomy of Tasks (§5)
Arithmetic
CoT (Wei et al., 2022b), Self-C (Wang et al., 2022e), Least-to-Most (Zhou et al., 2023), ZeroCoT (Kojima et al., 2022), Auto-CoT (Zhang et al., 2023b), LMSI (Huang et al., 2022), PAL (Gao et al., 2022), PoT (Chen et al., 2022b), Fine-tune-CoT (Ho et al., 2022)
Commonsense
CoT (Wei et al., 2022b), GenKnow (Liu et al., 2022c), Self-C (Wang et al., 2022e), Calibrator (Ye and Durrett, 2022), ZeroCoT (Kojima et al., 2022), Auto-CoT (Zhang et al., 2023b), COCOGEN (Madaan et al., 2022), LMSI (Huang et al., 2022), PINTO (Wang et al., 2023), RR (He et al., 2023)
Logical
Faithful (Creswell and Shanahan, 2022), LMLP (Zhang et al., 2022), Self-V (Weng et al., 2022), LAMBADA (Kazemi et al., 2022)
Symbolic
CoT (Wei et al., 2022b), Self-C (Wang et al., 2022e), Least-to-Most (Zhou et al., 2023), ZeroCoT (Kojima et al., 2022), PAL (Gao et al., 2022)
Multimodal
MarT (Zhang et al., 2023a), Multimodal-CoT (Zhang et al., 2023c), KOSMOS-1 (Huang et al., 2023), Visual-ChatGPT (Wu et al., 2023)

Figure 4.4.1: Taxonomy of Reasoning with Language Model Prompting [98]

1. **External Engines:** External engines serve as auxiliary tools that aid in the reasoning process. They can function as:

    - *Prompt Producers:* An example is a physical simulator used to generate structured prompts based on the context.

    - *Reasoning Executors:* Here, external engines help carry out the actual reasoning or problem-solving, as seen with code interpreters, which solve mathematical problems by executing code-like reasoning steps.

    - *Tool Extenders:* External tools or engines, such as calculators, spreadsheets, or database queries, can extend the model's abilities, improving reasoning and providing more accurate results.

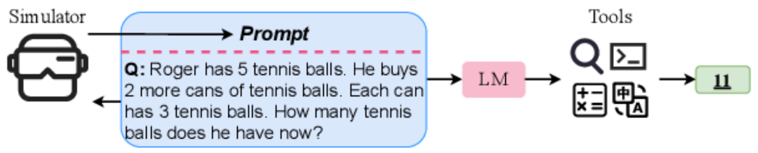

Simulator
Prompt
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
LM
Tools
11

Figure 4.4.2: External Engines for Reasoning enhancement [98]

2. **Process Optimization:** This approach refers to optimizing the reasoning process within the LLM itself. Techniques like Self-Consistency CoT, where multiple reasoning paths are explored and compared to identify the most consistent one, fall into this category. These methods aim to improve the internal logic and coherence of the model's reasoning process, as demonstrated in puzzle-solving tasks. This is further elaborated in Creswell et al. (2022) [22], where the Selection-Inference (SI) framework was proposed to tackle multi-step logical reasoning problems as shown in Figure 4.4.3. By alternating between selection and inference, this framework leverages LLMs as processing modules, leading to interpretable reasoning steps that enhance the trustworthiness of the system.

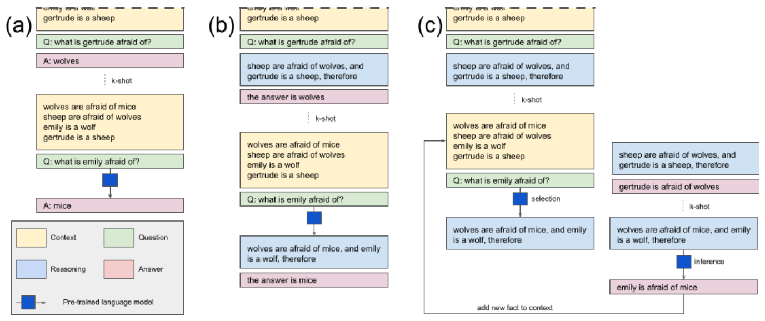

(a)
gertrude is a sheep
Q: what is gertrude afraid of?
A: wolves
k-shot
wolves are afraid of mice
sheep are afraid of wolves
emily is a wolf
gertrude is a sheep
Q: what is emily afraid of?
A: mice
Context
Question
Reasoning
Answer
Pre-trained language model
(b)
gertrude is a sheep
Q: what is gertrude afraid of?
sheep are afraid of wolves, and gertrude is a sheep, therefore
the answer is wolves
k-shot
wolves are afraid of mice
sheep are afraid of wolves
emily is a wolf
gertrude is a sheep
Q: what is emily afraid of?
wolves are afraid of mice, and emily is a wolf, therefore
the answer is mice
(c)
gertrude is a sheep
Q: what is gertrude afraid of?
sheep are afraid of wolves, and gertrude is a sheep, therefore
k-shot
wolves are afraid of mice
sheep are afraid of wolves
emily is a wolf
gertrude is a sheep
Q: what is emily afraid of?
selection
wolves are afraid of mice, and emily is a wolf, therefore
sheep are afraid of wolves, and gertrude is a sheep, therefore
gertrude is afraid of wolves
k-shot
wolves are afraid of mice, and emily is a wolf, therefore
inference
emily is afraid of mice
add new fact to context

Figure 4.4.3: Vanilla baseline (a) vs CoT (b) vs Selection-Inference (c) [22]

Knowledge Enhanced Reasoning:

Knowledge enhanced reasoning involves augmenting the LLM's reasoning capabilities by providing additional knowledge, either implicitly or explicitly. This can be achieved through:

1. **Implicit Knowledge:** In this case, the prompts are generated from the model's own internal knowledge. These prompts do not rely on external sources but rather leverage what the model has learned during pre-training. This method is often less reliable for domain-specific reasoning but is useful for general reasoning tasks, such as commonsense reasoning.

2. **Explicit Reasoning:** Prompts are retrieved or generated from external corpora or databases to guide the model's reasoning process. This approach is more suitable for complex tasks requiring domain-specific knowledge. Explicitly guiding the LLM with external information can significantly improve reasoning accuracy, as seen in symbolic reasoning and mathematical problem-solving.

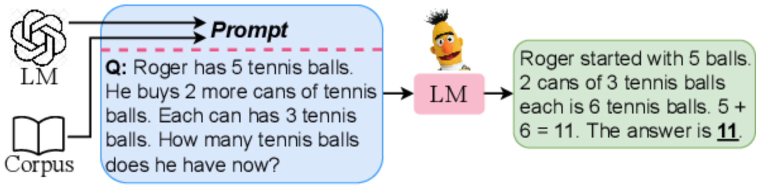

LM
Corpus
Prompt
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
LM
Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Figure 4.4.4: Knowledge Enhanced Reasoning [98]

4.4.2 Emergent Abilities of LLMs

An interesting phenomenon in LLMs is the concept of emergent abilities, a term that refers to capabilities that only appear when models reach a certain scale [126]. These abilities, such as advanced multi-step reasoning or program synthesis, are not predictable from the performance of smaller models. Instead, they emerge abruptly as the number of parameters increases, implying that LLMs may acquire complex reasoning abilities as a direct result of scale rather than fine-tuning or prompt engineering alone.

This phenomenon directly impacts the use of LLMs in reasoning tasks, as seen in models like GPT-3 and Gopher, where scaling up to billions of parameters resulted in improvements not only in text generation but also in complex logical inferences and puzzle-solving abilities. These emergent abilities highlight that scaling is a crucial factor in determining how well LLMs can perform on tasks that require deep logical reasoning.

In this thesis, we focus on models, which, although not as large as models that typically exhibit emergent abilities, are still substantial in size and capable of impressive reasoning tasks. A key objective of this work is to explore whether innovative prompting methods can enable these smaller yet powerful models to exhibit reasoning abilities that are usually associated with much larger models. This approach opens up the possibility of achieving similar emergent capabilities through efficient prompting, providing a more accessible and resource-effective alternative to scaling model size alone.

4.4.3 Challenges in Logical Reasoning with LLMs

Although LLMs have demonstrated the ability to perform well on tasks that require basic reasoning, multi-step logical reasoning remains a key challenge. Studies such as the one of Bao et al. (2023) [5] highlight these limitations. When LLMs encounter out-of-distribution (OOD) logical reasoning tasks—tasks that differ from those seen during training—their performance drops significantly. This occurs because models often fail to generalize beyond the patterns they have learned in their pre-training.

Furthermore, as discussed in Saparov et al. (2023) [105], LLMs struggle to handle more complex proofs in deductive reasoning when tested on a wide set of deduction rules, especially when the models are expected to generalize to proofs of greater complexity. This is particularly relevant in multi-step puzzle-solving tasks, where multiple logical steps must be chained together.

Out-of-Distribution Reasoning Challenges include:

- *Generalization to novel reasoning tasks:* LLMs tend to perform well on tasks they have been trained on, but when confronted with new logical structures or unfamiliar puzzle formats, their performance decreases.
- *Handling longer reasoning chains:* The ability to consistently generate and track multiple reasoning steps is one of the core difficulties for LLMs. This limits their performance on tasks that require depth in reasoning, such as the more complex types of puzzles and logical deductions.

In order to confront the above limitations Bao et al. (2023) [5] have proposed data augmentation techniques that perturb the training set, such as reshuffling options or replacing correct choices with alternative options, can improve the model's ability to generalize to new logical reasoning tasks. These techniques, when combined with fine-tuning and prompting, can lead to better OOD performance.

4.4.4 Comparison to Human Reasoning

Human reasoning and LLM-based reasoning differ significantly in key aspects such as flexibility, multi-step deduction, and the ability to manage uncertainty. Human reasoning is highly adaptable, allowing individuals to solve novel problems, infer missing information, and apply logical thinking to unfamiliar scenarios. In contrast, LLMs rely heavily on recognizing patterns from their training data. As presented, LLMs often struggle when confronted with out-of-distribution tasks that deviate from their learned examples, while humans can use intuition and past experience to address such challenges.

Another important distinction is in multi-step reasoning, where humans naturally excel at chaining together logical steps to reach conclusions. In complex puzzle-solving tasks, humans intuitively track and adjust their reasoning across multiple steps, something LLMs frequently struggle with. While LLMs can perform well on individual steps, their ability to integrate and execute multi-step logical deductions remains limited.

Additionally, humans are skilled at reasoning under uncertainty, adjusting their thinking when faced with ambiguity or incomplete information. LLMs, however, often exhibit confidence in their outputs, even when those outputs are based on incomplete or faulty logic, leading to issues like hallucinations, which are plausible yet inaccurate responses.

Chapter 5

Puzzle Solving and LLMs

This chapter presents an extensive exploration of puzzle-solving capabilities in Large Language Models (LLMs), expanding on the condensed work presented in our survey paper *Puzzle Solving using Large Language Models: A Survey* (Giadikiaroglou et al., 2024) [38]. We will provide an analytical and comprehensive discussion of puzzle types, methods, datasets, and benchmarks used in evaluating LLM reasoning within this domain.

Our work is rooted in the categorization of puzzles based on their structural differences and the cognitive skills they demand. Specifically, we introduce a clear distinction between rule-based puzzles and rule-less puzzles—a differentiation that reflects the varied knowledge demands required to tackle them effectively. This categorization is crucial for understanding how LLMs engage with different types of challenges and the reasoning methods that are most effective for each.

Furthermore, this chapter delves into the methodologies LLMs use to solve these puzzles, evaluating the effectiveness of different approaches such as prompting techniques, puzzle translation and fine-tuning. We also compare these methods with traditional problem-solving techniques, highlighting the strengths and limitations of both approaches in terms of cognitive processing and task efficiency.

In addition to methods, we provide a detailed overview of the datasets, benchmarks, and tasks that are commonly used to evaluate LLMs' reasoning abilities in the context of puzzle-solving. Understanding these benchmarks is essential for measuring performance and for identifying the gaps in current methodologies that might hinder further advancement in this area.

While LLMs have shown remarkable progress in handling certain reasoning tasks, they still face considerable challenges, like hallucinations [39, 46], particularly when dealing with more complex or abstract puzzles. These obstacles point to significant opportunities for future research. By focusing on these unsolved challenges, we aim to contribute to the development of more robust LLMs capable of tackling intricate reasoning

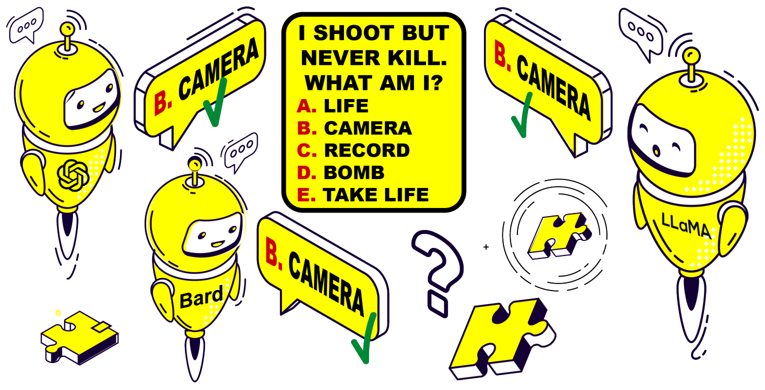


Figure 5.0.1: Riddle from RiddleSense [68]. GPT-4, LLaMA2-70B and Bard chose the right answer.

tasks with higher accuracy and consistency.

In this chapter, we explore the following key areas:

1. **Categorization of Puzzle Problems:** We introduce the distinction between rule-based and rule-less puzzles, illustrating how these categories require different problem-solving approaches.
2. **Methods and Strategies:** We analyze how LLMs tackle puzzles through various methods, such as prompting topologies and fine-tuning, and assess the effectiveness of each strategy across different puzzle types.
3. **Datasets, Benchmarks, and Tasks:** We present a comprehensive overview of the current benchmarks used to evaluate LLM reasoning and puzzle-solving performance.
4. **Comparison with Conventional Methods:** Finally, we compare LLM-based problem-solving approaches with conventional techniques, highlighting the advantages and limitations of each.

Our categorization diverges from existing logical reasoning taxonomies by emphasizing on the underlying cognitive processes and the skills required for puzzle solving, rather than the question format [76] or the nature of reasoning (deductive, inductive, abductive) [76, 135, 133, 98, 45, 32]. For instance, the existence of rules in puzzles such as Sudoku, Crosswords, or Minesweeper necessitates additional skills (e.g. strategy development) to correctly understand the game's rules or the ability to correctly format the output. In contrast, rule-less puzzles, such as riddles (Figure 5.0.1), programming challenges, and commonsense reasoning problems, leverage the model's inherent knowledge for solution derivation.

In our work, we define puzzles as problems that test cognitive abilities including logical reasoning, spatial cognition, and creative thinking by requiring the solver to discern patterns, apply deduction, and combine insights from available information in order to arrive at the correct solution. Additionally, puzzles that cannot be expressed through textual means—such as jigsaw puzzles [80] or those requiring multimodal understanding [37]—are excluded from our analysis, as are mathematical puzzles, which have been extensively covered in the recent work of Liu et al. (2023) [72].

The following sections will provide an in-depth analysis of each of these topics, giving a clearer understanding of the current state of puzzle-solving with LLMs and the potential future developments in this exciting field of research.

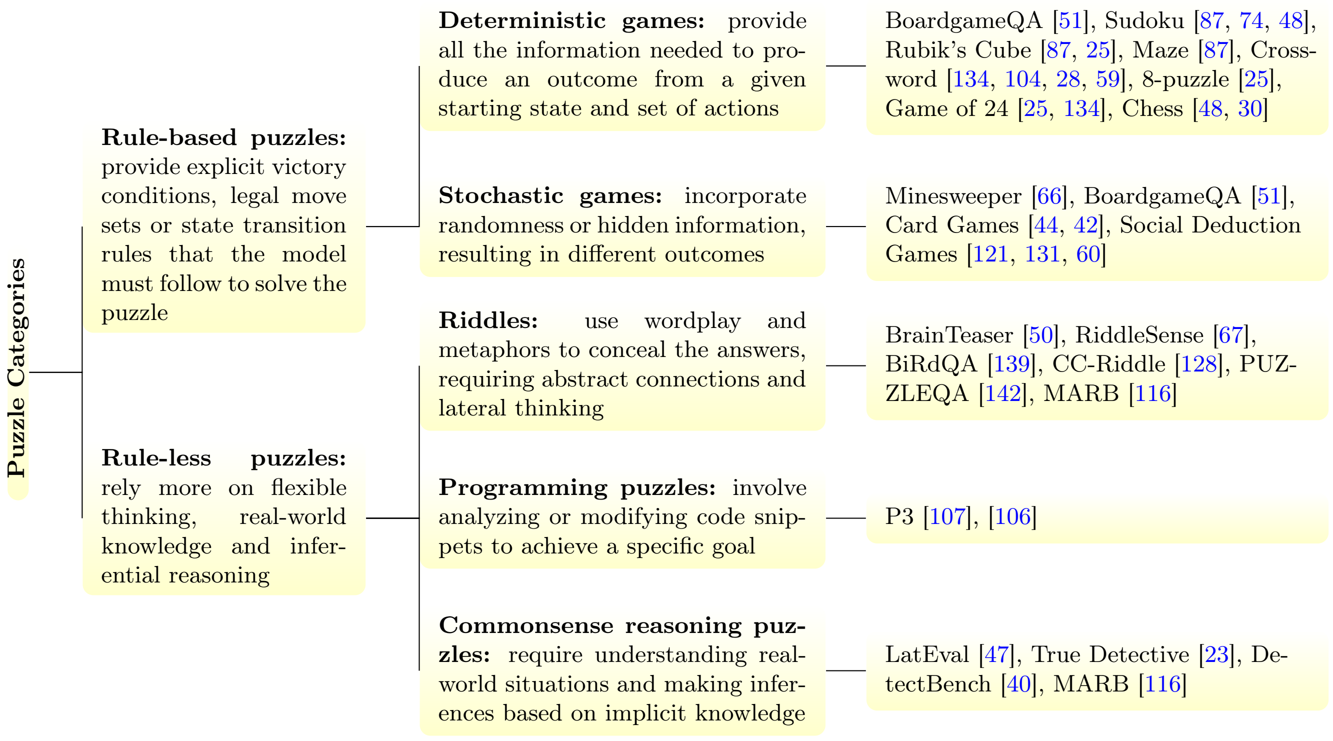
Contents

5.1 Categorization of Puzzle Problems

Understanding and assessing the reasoning capabilities of LLMs requires a systematic approach to classifying the types of puzzles they encounter. Puzzles, by their very nature, test a range of cognitive abilities, from logical deduction and strategic foresight to more abstract inferential reasoning. Therefore, categorizing these puzzles into distinct groups allows us to evaluate how LLMs engage with different cognitive challenges.

In this work, we distinguish puzzles based on their reliance on either formal rules or broader world knowledge combined with general inferential skills. This categorization, as illustrated in Figure 5.1.1, is crucial in highlighting the cognitive diversity inherent in puzzle-solving and the varied skill sets required to tackle them effectively. For instance, rule-based puzzles operate within clearly defined systems, often relying on logical deduction, strategic planning, and the strict application of formal rules. These puzzles demand the solver to operate within a closed environment, where the parameters and constraints are explicit, and the goal is to manipulate these rules to find the solution.

On the other hand, rule-less puzzles do not adhere to a strict set of rules but rather rely on a broader understanding of the world, practical knowledge, and inferential reasoning. These puzzles often pose open-ended problems that require LLMs to interpret a scenario, identify relevant clues, and synthesize information from their training data to arrive at a solution. This demands the model to go beyond the mere application of formal rules and instead engage in commonsense reasoning, using their learned knowledge to deduce, infer, or hypothesize the correct solution.


Puzzle Categories

Rule-based puzzles: provide explicit victory conditions, legal move sets or state transition rules that the model must follow to solve the puzzle

Deterministic games: provide all the information needed to produce an outcome from a given starting state and set of actions — BoardgameQA [51], Sudoku [87, 74, 48], Rubik's Cube [87, 25], Maze [87], Crossword [134, 104, 28, 59], 8-puzzle [25], Game of 24 [25, 134], Chess [48, 30]

Stochastic games: incorporate randomness or hidden information, resulting in different outcomes — Minesweeper [66], BoardgameQA [51], Card Games [44, 42], Social Deduction Games [121, 131, 60]

Rule-less puzzles: rely more on flexible thinking, real-world knowledge and inferential reasoning

Riddles: use wordplay and metaphors to conceal the answers, requiring abstract connections and lateral thinking — BrainTeaser [50], RiddleSense [67], BiRdQA [139], CC-Riddle [128], PUZZLEQA [142], MARB [116]

Programming puzzles: involve analyzing or modifying code snippets to achieve a specific goal — P3 [107], [106]

Commonsense reasoning puzzles: require understanding real-world situations and making inferences based on implicit knowledge — LatEval [47], True Detective [23], DetectBench [40], MARB [116]

Figure 5.1.1: A taxonomy of Puzzle Categories with the corresponding Datasets.

This division between rule-based and rule-less puzzles represents a significant shift in how we categorize reasoning challenges. Traditional logical reasoning taxonomies, such as those focusing on question formats or modes of reasoning (deductive, inductive, abductive), tend to emphasize the form of the puzzle or question rather than the underlying cognitive processes involved in solving it. By contrast, our approach categorizes puzzles by the skills required to solve them, offering a more cognitive-centric perspective on the reasoning challenges posed by different puzzles. This shift allows for a more nuanced analysis of how LLMs perform across a spectrum of reasoning tasks and highlights the areas where they excel or struggle.

In the following sections, we will delve deeper into these two broad categories, examining how LLMs approach each type of puzzle, the specific reasoning challenges they present, and the methods that have been developed to optimize LLM performance in solving these diverse puzzle types. By doing so, we aim to provide a comprehensive understanding of how LLMs engage with and solve puzzles, and what this reveals about their broader reasoning capabilities.

5.1.1 Rule-based Puzzles

Rule-based puzzles present a structured environment where formal rules govern every aspect of the problem, from legal actions to victory conditions. These puzzles require solvers to operate within clearly defined frameworks, often necessitating strategic planning and logical deduction to reach a solution. The key characteristic of rule-based puzzles is the presence of explicit state transition rules, where the outcome of a player's action is dictated by a fixed set of instructions. Given their structured nature, rule-based puzzles test an LLM's capacity for systematic reasoning and the ability to navigate within a constrained problem space.

This category can be further subdivided into two major types: deterministic puzzles, where each action leads to a predictable and consistent outcome, and stochastic puzzles, which incorporate elements of randomness or uncertainty. Understanding these subtypes is essential to evaluating how LLMs approach different types of structured environments, as the reasoning skills required for each differ considerably.

Deterministic Games

Deterministic games follow a strict cause-and-effect relationship between an action and its resulting state. In these games, given the current state and the player's action, the next state is always the same, adhering to the predefined rules without any randomness or ambiguity. Classic examples of deterministic puzzles include Chess, Sudoku, maze navigation, and the Rubik's Cube.

In Chess, for instance, moving a piece will always yield one specific, unambiguous board configuration depending on the current position and the applied move. Similarly, in Sudoku, the placement of a number in the grid follows fixed rules, leading to a unique solution if approached correctly. These puzzles require models to engage in forward search—evaluating future states based on current actions—while learning strategies that remain within the legal move set established by the rules. Success in deterministic games relies on the model's ability to fully understand the problem's constraints and apply logical reasoning to explore all possible outcomes and prune non-viable options from the search space.

Stochastic Games

In contrast, stochastic games introduce randomness or hidden information, meaning that the same player action can lead to different probability distributions over potential next states. Unlike deterministic puzzles, where each move results in a singular, predictable outcome, stochastic games require reasoning under uncertainty. The model must plan for multiple possible future states and manage risks associated with unknown variables.

Examples of stochastic puzzles include Minesweeper, where bomb locations are hidden, and Poker, where opponents' hands remain unknown. In Minesweeper, for instance, even though the rules governing cell interactions are fixed, the hidden bomb locations add an element of uncertainty, forcing the solver to balance logical deduction with probabilistic inference. Similarly, in Poker, a player's decision-making hinges on reasoning over incomplete information, as they must anticipate the possible hidden hands of opponents while weighing the risks and rewards of various strategies.

Mastering stochastic games requires models not only to operate within the bounds of formal rules but also to employ advanced probabilistic reasoning and risk management. LLMs attempting to solve these puzzles need to account for uncertainty by evaluating multiple potential outcomes, assigning probabilities to different scenarios, and selecting the action that maximizes the expected utility.

While both deterministic and stochastic games demand high levels of logical reasoning, the additional layer of unpredictability in stochastic puzzles poses a significant challenge. Whereas deterministic environments allow models to rely heavily on deduction and systematic forward search, stochastic settings require a blend of deductive logic, probabilistic inference, and decision-making under uncertainty. LLMs that excel in these environments must demonstrate a capacity for strategic foresight, risk analysis, and an ability to reason with incomplete information.

5.1.2 Rule-less Puzzles

Unlike rule-based puzzles that follow explicit guidelines and constraints, rule-less puzzles rely heavily on real-world knowledge, flexible thinking, and the ability to interpret vague or open-ended scenarios. These puzzles challenge the model to infer unspoken details, draw on broader conceptual knowledge, and employ creative reasoning to arrive at solutions. Rather than testing logical deduction within a structured environment, rule-less puzzles measure cognitive skills such as contextual interpretation, conceptual combination, and common-sense reasoning. The absence of formal rules or predefined moves requires LLMs to depend more on inference and their inherent knowledge of language and the world.

These puzzles fall into several distinct categories, each testing specific reasoning skills. The following subsections discuss some key types of rule-less puzzles: riddles, programming puzzles, and commonsense reasoning puzzles.

Riddles

Riddles are one of the most well-known forms of rule-less puzzles and are characterized by their use of clever wordplay, metaphors, and literary devices to obscure their solutions. The challenge in solving riddles lies in decoding the hidden meaning embedded within ambiguous or poetic language. Riddles often rely on figurative language or lateral thinking, requiring the solver to make abstract connections between seemingly unrelated concepts.

For example, the classic riddle, *"What gets wetter the more it dries?"*, conceals the solution, *"a towel"* through the metaphorical use of language. The trick is in understanding how the concepts of "getting wetter" and "drying" relate, not in a literal sense, but through an interpretation of how a towel absorbs water. Solving such puzzles tests an LLM's capacity for fluid reasoning, conceptual blending, and linguistic abstraction—all of which are critical for understanding and generating natural language that extends beyond basic syntactic rules.

Riddles assess a model's ability to move beyond conventional logic and embrace creativity and flexibility in reasoning. The ability of LLMs to handle such tasks depends heavily on their pre-existing knowledge of language, idioms, metaphors, and real-world phenomena, as well as their training on a wide range of textual data. Prompt engineering can enhance the model's ability to understand these more nuanced linguistic relationships by encouraging exploratory thinking, allowing LLMs to decode riddles more effectively.

Programming Puzzles

Programming puzzles may initially appear to be rule-bound, but they are classified as rule-less puzzles due to their reliance on understanding the abstract logic underlying code rather than adhering to formalized gameplay rules. In typical rule-based puzzles, such as Chess or Sudoku, the solver follows a strict set of predefined rules to manipulate game states toward a specific goal. In contrast, programming puzzles require the solver to reason about abstract operations and the behavior of code, which is often open to interpretation depending on the semantics of the programming language.

Unlike games with explicit victory conditions or state transitions, programming puzzles present code snippets or problems that require flexible thinking to solve. The rules in programming puzzles are not explicitly provided in the form of game mechanics but are instead embedded within the syntax and semantics of the code itself. As such, solving these puzzles requires the model to trace through potential execution paths, predict the behavior of the program, or identify logical errors based on its understanding of how programming constructs operate.

For instance, the following programming puzzle tests a solver's understanding of Python's integer division operator:

```
def mystery(x):
    return x // 2
print(mystery(10))
```

The model must predict the output based on its knowledge of Python's // operator, which performs integer division, yielding the result 5.

These puzzles can often be open-ended: multiple solutions may exist for a given problem, or the problem may not have a single, defined outcome, as opposed to deterministic puzzles with strict win conditions. The essence of programming puzzles is in debugging, modifying, and interpreting program behavior, which requires the solver to think flexibly and creatively, much like with riddles or commonsense reasoning puzzles.

Commonsense Reasoning Puzzles

Commonsense reasoning puzzles present everyday situations where certain details are omitted, requiring the solver to make plausible inferences based on general knowledge of the world. These puzzles challenge the model's ability to understand causal relationships, motivations, and unstated facts about familiar events, testing its capacity to reason through ambiguity by applying practical knowledge.

For example, the puzzle *"A man who was outside in the rain without an umbrella or hat didn't get a single hair on his head wet. Why?"* requires the solver to deduce that the man is bald. The puzzle deliberately withholds crucial information—namely, that the man has no hair—forcing the solver to infer this based on the unspoken implications of the scenario. Commonsense reasoning puzzles assess the model's ability to draw on its real-world knowledge and interpret unstated contextual cues, rather than simply relying on formal logic.

These puzzles highlight a key area where LLMs sometimes struggle: making the leap from general knowledge to specific inferences. While LLMs can generate text that sounds reasonable, accurately interpreting subtle or hidden details about real-world situations requires nuanced understanding, contextual reasoning, and the ability to "fill in the gaps." The challenge for LLMs is to determine which assumptions are valid based on the context provided and the likely background knowledge a human would apply to the situation.

5.2 Methods and Strategies

Puzzle solving presents a unique challenge, requiring a blend of cognitive abilities such as logical reasoning, inference-making, and abstract thinking. To tackle these challenges, a wide array of methods and strategies have been developed, each aimed at enhancing the problem-solving capacity of LLMs, particularly in complex and multi-step reasoning tasks.

In this section, we will explore the principal techniques employed in puzzle-solving tasks, focusing on their effectiveness within the specific context of reasoning with LLMs. While the literature on prompt engineering, model fine-tuning, and other techniques is vast and continues to expand rapidly [8, 13, 136, 18, 98, 71], we concentrate on the approaches most relevant to puzzle-solving. Rather than merely cataloging all available methods, we aim to provide an analytical review of how specific strategies can be harnessed to improve LLM performance in this domain. Our approach will focus on the following key methodologies: *Prompting Techniques*, *Puzzle Translation* strategies using neuro-symbolic methods, and *Fine-Tuning* for specialized domains and specific puzzle types.

Each method offers distinct advantages depending on the puzzle category (rule-based or rule-less), as outlined in earlier sections. For example, certain prompting techniques may enhance LLM reasoning in puzzles requiring strategic foresight, while others are more suited for abstract or commonsense reasoning tasks. Similarly, puzzle translation strategies aim to bridge the gap between human cognition and machine logic, translating complex puzzle structures into representations that LLMs can process more effectively. Fine-tuning, on the other hand, allows for domain-specific optimization, enabling LLMs to perform well in tasks like programming challenges or logical deduction puzzles by refining their model parameters with specialized datasets.

We provide a detailed overview of these techniques in the sections below, discussing their application to different puzzle categories. A comprehensive view of how these methods have been applied to specific puzzles is summarized in Table 5.1, illustrating the diversity of approaches used across various puzzle types. Additionally, we discuss how *conventional methods* for puzzle-solving, predating the LLM era, approached similar problems, and how these earlier strategies compare to current LLM-driven solutions.

Methods	**Rule-based Puzzles**		**Rule-less Puzzles**		
	Deterministic	**Stochastic**	**Riddles**	**Programming**	**Commonsense**
Prompting	-	-	-	-	-
Few-shot	✓	✓	✓	✓	✓
Chain-of-Thought	✓	✓	✓	✓	✓
Self-refine	✓				
Auto-CoT					✓
Complexity CoT					✓
Plan & Solve					✓
Detective Thinking					✓
Self-Consistency	✓				✓
Tree-of-Thoughts	✓				
Tree-of-uncertain-Thoughts	✓				
Inferential Exclusion Prompting			✓		✓
Graph-of-Thoughts	✓				
Everything-of-thoughts	✓				
Hints			✓		✓
Introduction/Summarization	✓	✓	✓	✓	✓
Puzzle Translation	-	-	-	-	-
Logic	✓				
Code					
Fine-Tuning	✓	✓	✓	✓	✓

Table 5.1: Methods used by each category of our taxonomy based on the puzzle benchmarks we collected

Table 5.1 delineates the various methods leveraged for puzzle-solving based on the datasets we have collected, illustrating the landscape of current LLM research in this domain. It particularly highlights the extensive methods applied to rule-based deterministic and rule-less commonsense puzzles. The absence of neuro-symbolic techniques and selection inference prompting indicates potential areas for expansion, especially considering their prospective benefits for LLMs grounded in logical reasoning datasets. The table further reflects the adaptability of certain methods like Chain-of-Thought, few-shot learning and fine-tuning, which are utilized across multiple puzzle types, hinting at their effectiveness. Based on this information, we not only catalogue the current state of method applications in puzzle-solving with LLMs but also highlight opportunities for innovative research in areas yet to be explored.

5.2.1 Prompting Methods

Prompting methods are at the heart of enhancing the puzzle-solving capabilities of LLMs. Through prompt design and manipulation, LLMs can be guided toward solving complex problems by structuring the input in ways that provide intermediate reasoning steps or multiple approaches to solving a given task. As discussed in Section §4.3, the fundamental principle behind prompting is to nudge the model's reasoning process by carefully selecting examples or structuring queries to facilitate better output. For puzzle solving, where logical deduction, strategy, or flexible thinking is required, these prompting methods offer unique solutions to both rule-based and rule-less puzzles.

The few-shot in-context learning paradigm, widely used in recent LLM applications, presents one or more demonstrations in the prompt to show the model how to solve a task [11, 26, 144]. This method has proven particularly effective across different puzzle types, both rule-based and rule-less, as it highlights the reasoning steps the model should emulate, significantly boosting performance without additional fine-tuning. By providing structured demonstrations, the model can generalize from the examples within the prompt, which is critical in solving diverse puzzle problems where the solution involves multiple reasoning steps or different types of abstraction. Interestingly, as Panagiotopoulos et al. (2024) [92] highlighted with their method (*RISCORE*), few-shot examples created with contextually reconstructed sentence-based puzzles in conjunction with the original examples show some of the best results compared to random few-shot examples.

Recent research has expanded on this idea by introducing a variety of thought structures—complex prompting frameworks that aim to improve puzzle-solving capabilities through strategic reasoning steps. These structures are designed to help LLMs break down complex puzzles into smaller, more manageable components. This section follows the work of Besta et al. (2024) [8] delving into three prominent topology-based prompting strategies—chain, tree, and graph topologies—each presenting unique advantages for solving puzzles and improving LLM performance in logical reasoning tasks.

Chain Topologies

The most notable approach in this category is **Chain-of-Thought (CoT)** prompting [127, 53]. CoT prompts the model to generate intermediate reasoning steps, building a logical chain of thoughts that leads to the final answer. CoT prompting follows two main approaches. The first involves a straightforward prompt like "Let's think step by step" to encourage stepwise reasoning before answering [53]. The second method provides a series of manual demonstrations, each consisting of a question and a reasoning process that leads to the answer [127]. The success of the second approach largely depends on carefully crafting these task-specific demonstrations individually. This approach has been widely adopted across both rule-based and rule-less puzzles, from simple logic puzzles to commonsense reasoning challenges. In solving puzzles, such as Sudoku or detective-style inference problems, CoT offers a step-by-step guide to reasoning through possible solutions. For instance, CoT has shown a considerable performance improvement over simple input-output (IO) prompts when applied to various puzzle types, such as riddles and deterministic games like Sudoku.

Other methods expand on CoT. For example, **Self-Refine** [79] enhances CoT by allowing the model to iteratively refine its reasoning steps, rethinking previous stages in light of new information. This technique was used successfully in solving the Game of 24, a rule-based/deterministic puzzle, and showed a 13% increase in success rate over standard CoT . Similarly, **Automatic CoT** [140] autonomously generates diverse reasoning chains, particularly effective in rule-less puzzles such as detective-style problems [40].

Other variants of CoT, like **Complexity CoT** [33] , guide the model towards generating more intricate reasoning steps by incorporating complex problem structures. This method has been shown to improve performance in puzzles that require deep, multi-step reasoning by selecting more elaborate and detailed outcomes. The **Plan-and-Solve (PS)** method [120] uses two prompts—one for generating an intermediate reasoning process and the corresponding answer and another for extracting the final answer from the generated reasoning steps.

Despite the varied approaches, none of these methods clearly outperformed CoT across all tested LLMs.

The **Detective Thinking Prompt** [40] builds on CoT-like approaches by asking the model to analyze multiple clues and synthesize this information into a logical conclusion. This method is especially effective in solving puzzles that require synthesizing disparate pieces of information, such as detective challenges. This method has achieved the best results in this study, with a 61.6% accuracy rate using GPT-4, highlighting that there is room for improvement.

Tree Topologies

Tree-based prompting strategies represent a more flexible and expansive method for guiding LLMs through puzzle-solving tasks. Unlike the linear, step-by-step process in chain topologies, tree topologies enable models to explore multiple reasoning paths in parallel, creating a structure where each possible solution can branch off from various intermediate steps. This broader exploration allows models to traverse potential outcomes and make decisions based on the most consistent or probable path, thus increasing the likelihood of arriving

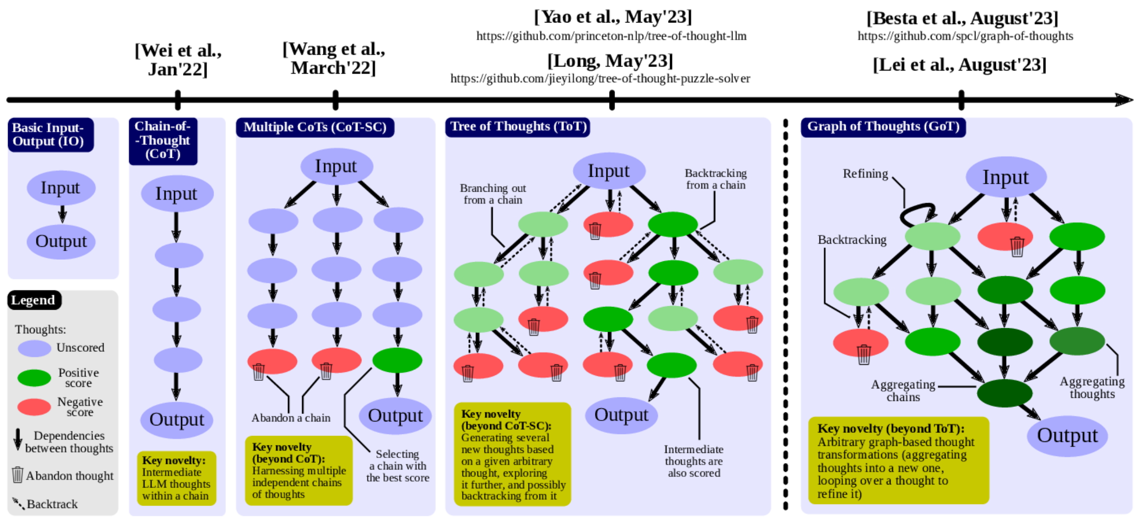

[Wei et al., Jan'22]
[Wang et al., March'22]
[Yao et al., May'23]
https://github.com/princeton-nlp/tree-of-thought-llm
[Long, May'23]
https://github.com/jieyilong/tree-of-thought-puzzle-solver
[Besta et al., August'23]
https://github.com/spcl/graph-of-thoughts
[Lei et al., August'23]
Basic Input-Output (IO)
Chain-of--Thought (CoT)
Multiple CoTs (CoT-SC)
Tree of Thoughts (ToT)
Graph of Thoughts (GoT)
Input
Output
Legend
Thoughts:
Unscored
Positive score
Negative score
Dependencies between thoughts
Abandon thought
Backtrack
Key novelty: Intermediate LLM thoughts within a chain
Key novelty (beyond CoT): Harnessing multiple independent chains of thoughts
Abandon a chain
Selecting a chain with the best score
Branching out from a chain
Backtracking from a chain
Key novelty (beyond CoT-SC): Generating several new thoughts based on a given arbitrary thought, exploring it further, and possibly backtracking from it
Intermediate thoughts are also scored
Refining
Backtracking
Aggregating chains
Aggregating thoughts
Key novelty (beyond ToT): Arbitrary graph-based thought transformations (aggregating thoughts into a new one, looping over a thought to refine it)

Figure 5.2.1: Evolution of reasoning topologies used in prompting schemes [8].

at the correct solution. Tree-based approaches have been especially impactful in rule-based puzzles, but their utility in rule-less puzzles has shown varied results.

One of the key methods leveraging tree structures is **Self-Consistency (SC)**, introduced by Wang et al. (2022) [123]. Self-Consistency modifies the traditional Chain-of-Thought (CoT) prompting by generating multiple reasoning paths for a single problem. Instead of producing one linear sequence of reasoning, SC allows the model to generate several candidate chains and then selects the most consistent outcome across these multiple reasoning paths. This technique has been tested extensively on rule-based/deterministic puzzles such as the 8-puzzle, Game of 24, and Pocket Cube, as well as rule-less commonsense reasoning tasks.

In deterministic puzzles, SC has consistently shown a slight but meaningful performance gain over CoT. For example, in the Game of 24, SC improved success rates by a small but statistically significant margin, demonstrating the advantage of reasoning over multiple paths when dealing with complex state transitions [134]. However, in rule-less puzzles such as commonsense reasoning challenges, SC has not demonstrated the same level of improvement [40]. This discrepancy may arise from the inherent difference in how LLMs handle uncertainty versus structured logic. While deterministic puzzles benefit from the precise nature of SC, the fluid and often abstract nature of rule-less puzzles makes it harder to consistently apply this method to improve outcomes.

The **Tree-of-Thought(s) (ToT)** approach, proposed by two different studies [134, 74], expands on the idea of generating multiple reasoning paths by constructing a full decision tree of possible solutions. ToT has been predominantly applied to rule-based/deterministic puzzles, where it has demonstrated significant success. Each branch in the tree represents a possible decision or action the model could take, allowing it to consider a broader range of outcomes than linear approaches like CoT. By systematically exploring multiple branches, the model can navigate through the puzzle space more effectively.

ToT has shown remarkable performance increases over CoT. Depending on the puzzle type and the depth of the tree, success rates have improved by as much as 26% [86] to 70% [134]. This parallel exploration, despite the increased computational overhead due to the higher number of model invocations, yields significant improvements in accuracy and robustness [25]. ToT's depth control allows it to adjust how deeply the model explores potential solutions, balancing performance with computational efficiency.

An extension of ToT, the **Tree-of-Uncertain-Thought (TouT)** method [86], incorporates uncertainty into the reasoning process. While ToT assumes that the puzzle can be solved by exploring deterministic paths, TouT allows the model to factor in ambiguity or incomplete information at each decision node. This is particularly useful in puzzles that present incomplete or probabilistic information, such as those where the outcome of an action is uncertain until executed, but the method is not still tested on puzzles of that category.

In comparative tests, TouT outperformed ToT on the same rule-based/deterministic puzzles, achieving a 9%

higher success rate in the Game of 24 and a 3% improvement on mini-crosswords [86]. By incorporating uncertainty into the tree structure, TouT provides a more nuanced approach to puzzles where risk management and probabilistic reasoning are necessary. This capability makes it a more flexible option than ToT, especially in environments where not all variables are known at the outset or where multiple possible outcomes need to be evaluated concurrently. Thus, it would be an interesting area of research to apply this method to stochastic puzzles as well.

While the previous methods have primarily focused on deterministic puzzles, the **Inference-Exclusion-Prompting (IEP)** method [116], demonstrates how tree-based structures can be adapted for rule-less puzzles. IEP employs a combination of forward and backward reasoning to approximate human logical reasoning, where the model alternates between forward steps of reasoning and backward elimination of improbable options. This iterative process mimics human deductive logic, particularly in riddles or commonsense reasoning tasks where multiple potential solutions must be considered and excluded until the correct one emerges.

When combined with CoT, IEP has achieved some of the best results in riddles and commonsense puzzles, reaching an 82% success rate on puzzles (compared to 81% with zero-shot CoT) and 79% on riddles (compared to 82% with zero-shot CoT) [116]. These results demonstrate that by leveraging tree-based reasoning and exclusion strategies, IEP enhances the model's ability to navigate ambiguous problem spaces and make informed decisions based on both positive and negative evidence.

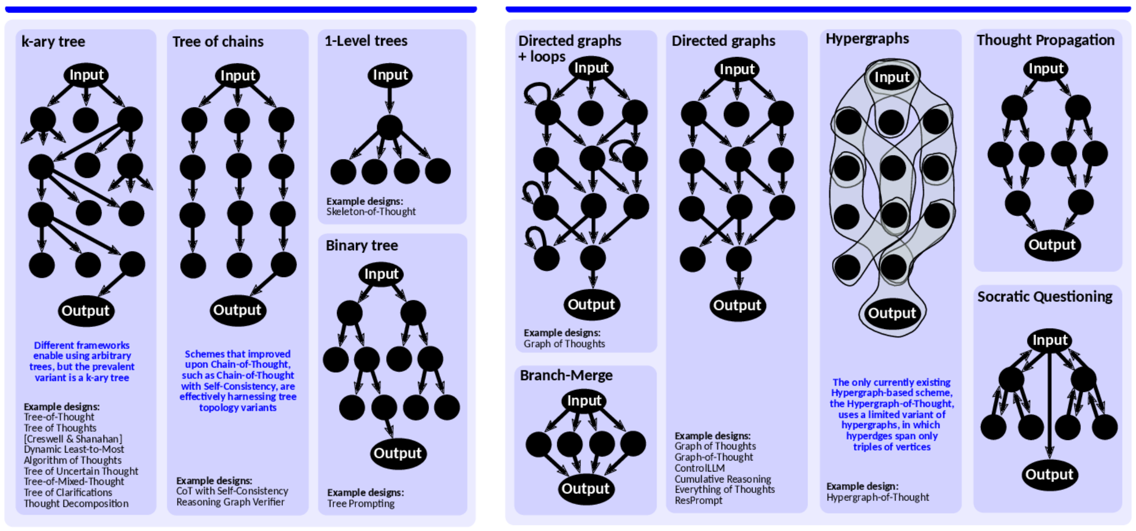


Figure 5.2.2: Variants of tree and graph prompting topologies [8].

Graph Topologies

Graph-based prompting methods extend the flexibility of tree-based approaches by allowing for more complex interconnections between reasoning steps. While tree structures rely on strictly hierarchical relationships, graph structures permit more general connectivity, enabling multiple paths and cycles that can explore a broader range of reasoning possibilities. These methods are especially useful for tasks that require revisiting earlier decisions, considering alternative pathways, and combining multiple lines of reasoning into a cohesive solution.

The **Graph-of-Thought(s) (GoT)** method [7, 62], applies graph-based reasoning to the puzzle-solving domain. In GoT, the reasoning process is represented as a graph, where nodes correspond to intermediate reasoning states or steps, and edges represent possible transitions between these states. This structure allows the model to explore multiple reasoning paths in parallel and return to previously explored nodes, forming a network of potential solutions rather than a linear or tree-like sequence.

GoT has been tested primarily on rule-based/deterministic puzzles. However, the results on some puzzles have shown that GoT underperforms compared to other prompting methods, such as Tree-of-Thought (ToT) [25]. In particular, GoT demonstrated a performance decrease ranging from 2% to 6% compared to ToT in Game of 24, 8-Puzzle and Pocket Cube. This lower performance is attributed to the higher complexity of managing multiple interconnected paths, which can lead to increased ambiguity in the decision-making process. While GoT offers greater flexibility in exploring reasoning steps, this comes at the cost of increased computational overhead and potentially less precise reasoning when dealing with well-structured puzzles that benefit from more focused exploration.

Building upon the concepts of tree and graph-based reasoning, **Everything-of-Thought (XoT)** [25], represents the most advanced graph-based approach currently applied to puzzle solving with LLMs. XoT integrates Monte Carlo Tree Search (MCTS) with LLM-based reasoning to enhance the exploration of possible solutions. MCTS, a well-established technique in AI for decision-making in uncertain environments (such as Go and Chess), allows the model to simulate multiple potential outcomes and select the most promising path based on probabilistic estimates.

XoT's combination of MCTS with LLMs has proven highly effective in rule-based/deterministic puzzles, where structured, probabilistic exploration can significantly improve performance. For example, XoT achieved improvements in success rates ranging from 53% to 69% compared to ToT, depending on the puzzle complexity and depth of reasoning involved. This improvement is largely due to MCTS's ability to effectively navigate the solution space by balancing exploration (trying out different reasoning paths) and exploitation (focusing on the most promising solutions).

Another key advantage of XoT is its efficiency in LLM invocations. Compared to other methods like CoT, SC, and ToT, XoT requires fewer model invocations to reach a solution. This reduction in computational load is particularly valuable in scenarios where multiple invocations of the model are expensive or time-consuming. XoT's integration of graph-based reasoning and probabilistic search makes it the most robust and resource-efficient method currently available for rule-based puzzle solving.

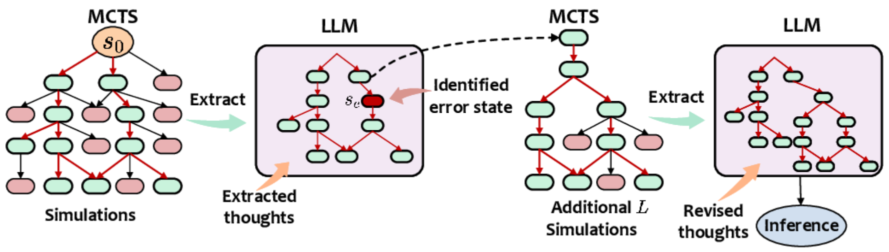


Figure 5.2.3: An illustration of thought revision process in XOT [25].

Additional Prompting Techniques

Beyond the prompting methods falling under the previous topologies, other techniques that incorporate supplementary information have been explored in the context of puzzle solving with LLMs. For instance, the use of hints in riddles and commonsense puzzles has been shown to improve performance by providing the model with additional context or clues. However, the effectiveness of hints can vary depending on the nature of the puzzle and the language involved. For example, Zhang et al. (2021) [139] observed that while hints generally improve performance on English riddles of their dataset (BiRdQA), they can lead to worse results on Chinese riddles, where linguistic and cultural nuances play a significant role in puzzle comprehension.

In addition to hints, the use of introductions and summarizations has been explored to provide the model with a broader context or to focus its attention on the key aspects of a puzzle. These techniques have generally yielded positive results, but their effectiveness is highly dependent on the puzzle type and the clarity of the additional information provided. In some cases, overly detailed introductions can distract the model from the core reasoning task, while well-targeted summarizations can help the model focus on relevant clues and improve accuracy.

5.2.2 Puzzle Translation

In puzzle solving, one of the core challenges for Large Language Models (LLMs) is translating puzzles, expressed in natural language, into representations that can be more easily processed by symbolic solvers or other external tools. Unlike approaches that assess the LLM's capacity to solve puzzles directly, puzzle translation focuses on the model's ability to encode the puzzle into an appropriate format—often logical rules or executable code—that external engines or solvers can then use to find the solution. This process is central to neuro-symbolic reasoning, which combines the flexibility of neural networks (like LLMs) with the precision of symbolic reasoning methods.

Neuro-Symbolic Techniques for Logic Rule Translation

A key approach in neuro-symbolic reasoning involves converting natural language puzzles into formal logic representations, which are then solved by symbolic reasoning systems.

Ishay et al. (2023) [48], demonstrate the power of this approach by using models like GPT-3 and GPT-4 to transform natural language descriptions of puzzles into logic rules written in formal languages, such as Answer Set Programming (ASP). For instance, in logic puzzles like Sudoku or chess, LLMs are prompted to generate predicates and rules that formally encode the puzzle's constraints and victory conditions. Once translated into this logical form, a symbolic solver can quickly compute the solution. The study showed remarkable results: GPT-4 achieved 92% accuracy on a logic puzzle dataset [85], significantly outperforming its few-shot (7%) and zero-shot (21%) baselines on the same dataset. They note that in few-shot settings, LLMs can generate complex programs that humans can easily refine and correct in case of code errors.

Other frameworks, such as Logic-LM [90], LINC [88], and a methods proposed by Yang et al. (2023) [132], have also demonstrated success in applying neuro-symbolic techniques to logical reasoning tasks. These frameworks primarily focus on translating reasoning tasks into formal rule-based representations that can be interpreted by symbolic engines, showcasing their potential in domains that require precise, logical inference. However, it is important to note that these techniques, while promising, have yet to be fully explored in the specific domain of puzzle solving. Most research to date has concentrated on broader logical reasoning tasks rather than on the highly structured nature of puzzles.

Puzzle Translation into Code

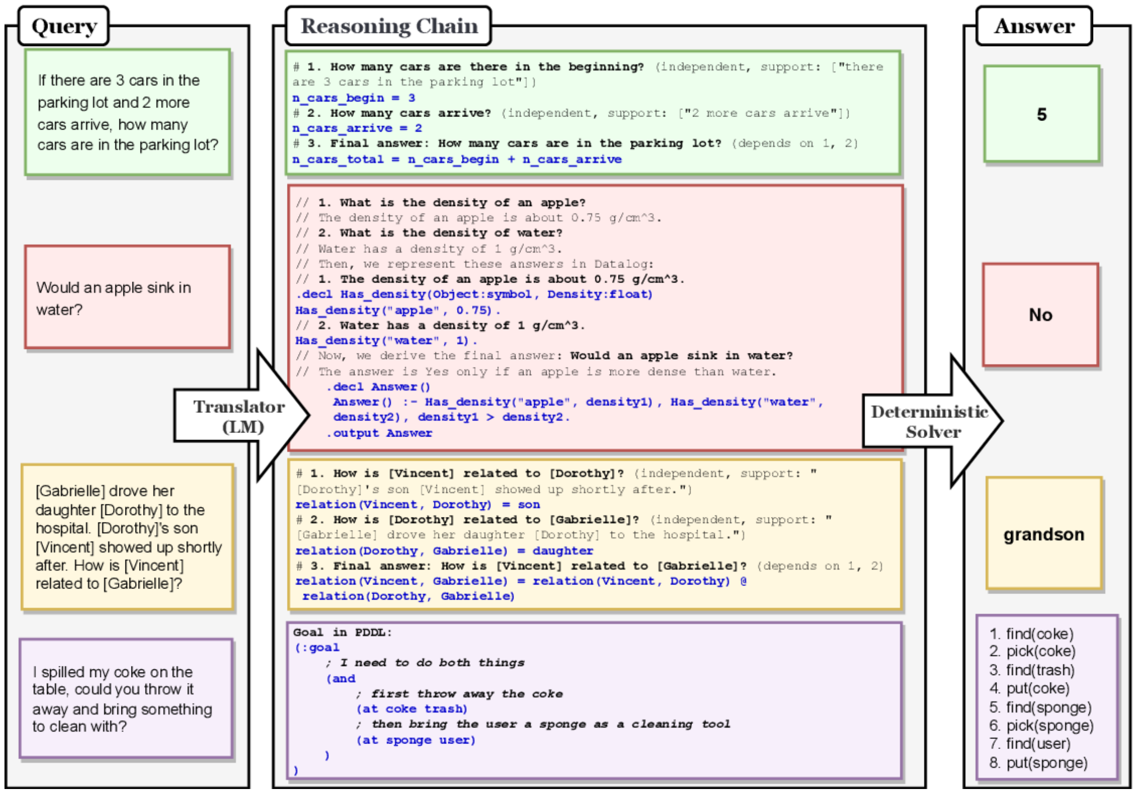
While much of the current research focuses on translating puzzles into logic rules, there is a growing interest in using LLMs to generate executable code for puzzle solutions. Code-based approaches, such as Program of Thought (PoT) prompting [16], Program-Aided Language (PAL) [35] and Faithful CoT [78], have already shown significant promise in solving logical and mathematical reasoning tasks by converting the reasoning process into Python code. These methods are particularly effective when dealing with tasks that require step-by-step computational reasoning, where the model generates code that mimics the logic of the puzzle solution. For instance, in PoT, the model is prompted to create a program that simulates the logic required to solve a problem, allowing for a more structured and explicit reasoning process.

Although PoT, PAL and Faithful CoT have primarily been applied to mathematical reasoning datasets, there is considerable potential to adapt these methods to puzzle-solving tasks. By translating the puzzle's reasoning process into executable code, LLMs could leverage external programming environments to handle complex puzzles that require iterative or computational steps, such as maze-solving or algorithmic puzzles. However, this remains an area of active research, and so far, there have been no comprehensive studies focused on translating puzzles directly into code.

Suitability for Rule-Based Puzzles

Given the structured nature of rule-based puzzles—where explicit rules and well-defined victory conditions govern the puzzle space—neuro-symbolic approaches and code generation methods are naturally well-suited for these problems. The ability to encode rule-based puzzles into logic rules or code allows models to bypass the need for complex internal reasoning, offloading much of the problem-solving process onto symbolic solvers or programming environments. As a result, rule-based puzzles have become the primary focus for puzzle translation research, while rule-less puzzles, which lack formal structures or consistent rules, present unique challenges that make translation more difficult. Currently, there have been no significant efforts to apply neuro-symbolic techniques or code translation methods to rule-less puzzles, as these puzzles typically require more general inferential skills and are less amenable to formalization.

Nevertheless, the exploration of translation methods for rule-based puzzles offers valuable insights into how LLMs can interface with symbolic reasoning systems and external engines, pointing to future possibilities for integrating neural and symbolic approaches in complex reasoning tasks.


Query
Reasoning Chain
Answer
Translator (LM)
Deterministic Solver
If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot?
Would an apple sink in water?
[Gabrielle] drove her daughter [Dorothy] to the hospital. [Dorothy]'s son [Vincent] showed up shortly after. How is [Vincent] related to [Gabrielle]?
I spilled my coke on the table, could you throw it away and bring something to clean with?
5
No
grandson
1. find(coke)
2. pick(coke)
3. find(trash)
4. put(coke)
5. find(sponge)
6. pick(sponge)
7. find(user)
8. put(sponge)

```
# 1. How many cars are there in the beginning? (independent, support: ["there
are 3 cars in the parking lot"])
n_cars_begin = 3
# 2. How many cars arrive? (independent, support: ["2 more cars arrive"])
n_cars_arrive = 2
# 3. Final answer: How many cars are in the parking lot? (depends on 1, 2)
n_cars_total = n_cars_begin + n_cars_arrive
```

```
// 1. What is the density of an apple?
// The density of an apple is about 0.75 g/cm^3.
// 2. What is the density of water?
// Water has a density of 1 g/cm^3.
// Then, we represent these answers in Datalog:
// 1. The density of an apple is about 0.75 g/cm^3.
.decl Has_density(Object:symbol, Density:float)
Has_density("apple", 0.75).
// 2. Water has a density of 1 g/cm^3.
Has_density("water", 1).
// Now, we derive the final answer: Would an apple sink in water?
// The answer is Yes only if an apple is more dense than water.
    .decl Answer()
     Answer() :- Has_density("apple", density1), Has_density("water",
     density2), density1 > density2.
    .output Answer
```

```
# 1. How is [Vincent] related to [Dorothy]? (independent, support: "
[Dorothy]'s son [Vincent] showed up shortly after.")
relation(Vincent, Dorothy) = son
# 2. How is [Dorothy] related to [Gabrielle]? (independent, support: "
[Gabrielle] drove her daughter [Dorothy] to the hospital.")
relation(Dorothy, Gabrielle) = daughter
# 3. Final answer: How is [Vincent] related to [Gabrielle]? (depends on 1, 2)
relation(Vincent, Gabrielle) = relation(Vincent, Dorothy) @
 relation(Dorothy, Gabrielle)
```

```
Goal in PDDL:
(:goal
    ; I need to do both things
    (and
        ; first throw away the coke
        (at coke trash)
        ; then bring the user a sponge as a cleaning tool
        (at sponge user)
    )
)
```

Figure 5.2.4: Examples from various tasks applying the Faithful-CoT method [78].

5.2.3 Fine-Tuning

Fine-tuning LLMs has emerged as a powerful strategy for enhancing their reasoning capabilities, especially when tasked with solving complex puzzles. By refining models on domain-specific data, researchers aim to improve the performance of LLMs across a variety of puzzle types, ranging from general logical reasoning tasks to more specific, rule-based, and rule-less puzzles. Fine-tuning can adapt a model's general knowledge to solve domain-specific challenges, thus extending its capabilities beyond what it can achieve with mere prompting.

Logical Reasoning

In the domain of logical reasoning, fine-tuning has demonstrated notable success by enhancing the LLM's ability to emulate the precise, step-by-step nature of symbolic reasoning. One prominent example is LoGiPT [29], a language model fine-tuned specifically for logical reasoning tasks. LoGiPT incorporates a unique instruction-tuning dataset comprising natural language (NL) logical questions paired with symbolic reasoning steps. The model is fine-tuned to bypass common syntax errors encountered when parsing NL into symbolic languages, allowing it to directly produce answers without relying on external tools. This approach has made LoGiPT a formidable tool for tasks that require the integration of symbolic logic within the broader scope of LLM reasoning.

Similarly, LogiT5 [75] leverages a multi-task learning approach to improve reasoning capabilities across a range of logical domains. By fine-tuning on the LOGIGLUE benchmark, which consists of diverse logical reasoning datasets, LogiT5 is able to generalize well across different tasks, particularly in scenarios with limited training data. The model's ability to transfer knowledge from one task to another enhances its performance across various logical reasoning challenges, demonstrating the power of multi-task fine-tuning for expanding the scope of LLM reasoning abilities.

Rule-Based Puzzles

When applied to rule-based puzzles, fine-tuning has produced mixed results. Rule-based puzzles, such as Sudoku, Rubik's Cube, and mazes, require the model to understand and operate within fixed, formal rules. In these puzzles, the model must systematically explore possible states, actions, and outcomes to arrive at a correct solution.

A study by Noever et al. (2021) [87] highlights the challenges of fine-tuning models like GPT-2 on rule-based deterministic puzzles. Despite fine-tuning on Sudoku and Rubik's Cube puzzles, the models achieved suboptimal results. The study suggests that the brief fine-tuning period and limited training examples were insufficient to significantly improve the model's performance in these highly structured domains. The complexity of these puzzles often demands a deep understanding of game strategies, such as minimax search or dynamic programming, which fine-tuning alone might not fully capture.

In contrast, fine-tuning has yielded more encouraging results in specific areas of rule-based puzzles. For instance, studies on crosswords [104, 28], show that fine-tuned LLMs can, in some cases, outperform traditional non-neural baselines, although cryptic crosswords still present significant challenges. Fine-tuning has also proven effective when combined with CoT reasoning and proof generation [51]. This combination produced some of the best results in board game puzzles, showcasing how targeted fine-tuning can enhance performance in well-structured, rule-based environments.

Rule-Less Puzzles

Fine-tuning also plays a key role in improving LLM performance on rule-less puzzles, where models must rely on flexible, real-world knowledge and inferential reasoning. Riddles, which often employ wordplay and clever misdirection, benefit from fine-tuning on datasets that emphasize commonsense knowledge and conceptual blending.

For example, the study by Lin et al. (2021) [67] illustrates how models like BERT [24], RoBERTa [73], and ALBERT [61] perform better when fine-tuned on the RiddleSense dataset alongside CommonsenseQA [114], leveraging commonsense knowledge to interpret riddles more effectively. In particular, Zhang et al. (2021) [139] found that combining fine-tuning on ALBERT-XXL with transfer learning from CommonsenseQA resulted in a 4% accuracy improvement over models that only underwent fine-tuning, illustrating the benefits of transfer learning in riddles and commonsense puzzles.

Beyond riddles, fine-tuning has also shown success in other rule-less puzzle domains. In commonsense reasoning, fine-tuning allows models to develop a deeper understanding of everyday knowledge [23]. Similarly, fine-tuning has been applied to programming puzzles with promising results. In the domain of programming puzzles, models fine-tuned on datasets like Programming Puzzles [107] demonstrate improved accuracy in solving code-based challenges by refining their understanding of programming semantics and logical operations.

Overall, fine-tuning LLMs has demonstrated substantial improvements in both rule-based and rule-less puzzle-solving tasks. By tailoring the model's capabilities to specific domains and reasoning types, fine-tuning bridges the gap between general-purpose LLMs and the specialized demands of puzzles, enabling more precise and effective problem-solving across diverse puzzle categories.

5.2.4 Conventional Methods

AI and Machine Learning methods have long been applied to puzzles and games, with algorithms like Deep Blue [12] and AlphaZero [109] for Chess and Go, renowned for their exceptional results. This section contrasts "traditional" methods used to solve various puzzles with those derived from large language models (LLMs). Note that the aim of this paper isn't to determine the superior method for each puzzle, but to highlight the distinctive reasoning abilities of LLMs within diverse puzzle contexts. We particularly focus on rule-based puzzles, extensively addressed using conventional methods due to their structured, well-defined environments which require systematic strategies to achieve a solution. Conversely, rule-less puzzles such as riddles primarily test the logical, commonsense reasoning and creativity of models, without a clear path of steps to follow in order to find the solution, so we do not analyze this category.

Chi et al. (2013) [17] utilized three techniques to solve **Sudoku**: backtracking, simulated annealing, and alternating projections. The backtracking method, a brute-force depth-first search, consistently resolves puzzles across all difficulty levels, albeit slowly. Constraint programming transforms Sudoku into a constraint satisfaction problem, swiftly enforcing constraints to deduce solutions, often within milliseconds [110]. These methods always find a solution for Sudoku puzzle, in contrast with LLMs that have not achieved results better than 80% for 5x5 puzzles [74].

In their study on **Rubik's Cube**, Chen (2022) [14] employed several traditional methods including Korf's algorithm [55], which combines Iterative-Deepening Depth-First Search (IDDFS) with the A* algorithm and a heuristic search database. Both Thistlethwaite's [1] and Kociemba's [2] algorithms utilize group theory and similar search techniques to streamline the solving process, with Kociemba's version enhancing efficiency by simplifying the group structure. While all these algorithms effectively solve the Rubik's Cube—a task challenging for LLMs—Korf's method is particularly noted for its efficiency. Additionally, the study explored a machine learning strategy that integrates Monte-Carlo Tree Search (MCTS) with breadth-first search, yielding more optimized solutions, albeit at a lower efficiency. There have also been various attemts to solve Rubik's Cube using Reinforcement Learning (RL) like DeepCubeA [81, 1] and others [113], which although find a solution in relatively few steps are time-consuming, with duration varying from 38.7 to 75.6 seconds [113].

Mazes are puzzles that can be solved by applying simple algorithms like depth-first search, A* or Trémaux's algorithm. However these problems are good for testing the spatial reasoning of LLMs. RL has also been utilized to solve mazes with [6] leveraging LLM feedback during training.

MCTS has been used to solve **Game of 24**, **8-Puzzle** and **Pocket Cube**, achieving surpassing many LLM techniques, including CoT, CoT-SC, ToT and GoT [25]. Additionally, Rozner et al. (2021) [104] besides fine-tuning T5 for solving cryptic crosswords, have also used non-neural baselines including a WordNet-based heuristic model, a K-Nearest Neighbours bag of words model and a rule-based model, showing that the fine-tuning of T5 had the best results among them.

Finally, Studholme (2001) [111] proposed a method for solving **Minesweeper** by considering it as a constraint satisfaction problem (CSP). The core strategy involves transforming the game's challenges into a set of logical constraints that must be satisfied to avoid mines effectively.

In conclusion, most conventional methods used to solve rule-based puzzles employ deterministic approaches that reliably produce solutions, in stark contrast to the unpredictable nature of LLMs. Another advantage of these traditional methods is their explainability and interpretability, crucial attributes for thoroughly evaluating algorithms and understanding their decision-making processes. However, these methods can sometimes exhibit increased time complexity, indicating a potential trade-off between reliability and efficiency [113].

5.3 Datasets, Benchmarks and Tasks

The evaluation of LLMs in puzzle-solving requires the exploration of diverse datasets, benchmarks, and task formats to effectively assess their reasoning abilities. Datasets play a pivotal role in determining how well LLMs perform across different types of puzzles, providing structured environments for testing and comparing various methods and strategies. This section delves into the datasets and benchmarks used within the puzzle-solving domain, categorizing them according to our previously established taxonomy of rule-based and rule-less puzzles. Each category highlights the distinct reasoning challenges posed by the puzzles and the corresponding evaluation metrics employed to gauge LLM performance. By systematically analyzing these datasets, we can better understand the versatility of LLMs and the impact of various puzzle-solving techniques discussed in Section §5.2. This section serves as an essential foundation for understanding how LLMs are tested and benchmarked, offering insights into their reasoning capabilities across different puzzle formats.

[1] https://www.jaapsch.net/puzzles/thistle.htm
[2] https://kociemba.org/

5.3.1 Rule-Based Puzzles

Rule-based puzzles provide a structured, closed-world environment ideal for evaluating LLMs' logical reasoning and ability to operate within a set of predefined rules. These puzzles are divided into two subcategories: *deterministic* and *stochastic* puzzles. Deterministic puzzles, such as Sudoku, Rubik's Cube, Crosswords, and the 8-puzzle, adhere to fixed, unchanging rules where the same actions always lead to the same outcomes. These puzzles challenge LLMs to leverage strategies that rely on forward search and deductive reasoning within clearly defined parameters.

In contrast, stochastic puzzles like Minesweeper, card games, or social deduction games introduce an element of uncertainty. Here, the same action may lead to different outcomes due to hidden factors or randomized elements, requiring the model to reason over probabilistic outcomes and make decisions in the face of incomplete information. While much of the research has concentrated on deterministic puzzles—where success can be measured through logical consistency and rule adherence—there remains a substantial gap in addressing the challenges presented by stochastic puzzles. These environments demand reasoning capabilities that extend beyond deduction to include risk assessment and decision-making under uncertainty, marking a promising area for future research.

Deterministic Puzzles

Sudoku is one of the most prominent benchmarks for assessing LLMs' logical reasoning capabilities due to its structured complexity and reliance on deductive problem-solving. Noever et al. (2021) [87] conducted one of the first explorations by fine-tuning GPT-2 [100] on a massive dataset of 1 million Sudoku games, experimenting with different input representations. They used a compact, single-string format where empty cells are marked with "-", and claimed that a matrix representation of the puzzle might improve the model's learning efficiency by better capturing the spatial structure. Further research by found success using nested lists (e.g., [[3,*,*,2], [1,*,3,*],[*,1,*,3],[4,*,*,1]]) for puzzle representation, applying the Tree-of-Thought (ToT) method [74]. This approach significantly outperformed simple prompting techniques, particularly for smaller puzzles, highlighting the potential of reasoning-driven methods in deterministic puzzles. In a similar vein, Ishay et al. (2023) [48] used GPT-3 and GPT-4 to generate answer set programming (ASP) rules for Sudoku, as well as other puzzles like Jobs puzzles and logic problems, demonstrating that well-prompted LLMs could effectively encode logical rules and generate accurate ASP-based solutions. In fact, GPT-4 achieved a notable 92% accuracy in logic puzzle-solving, showing the effectiveness of neuro-symbolic approaches.

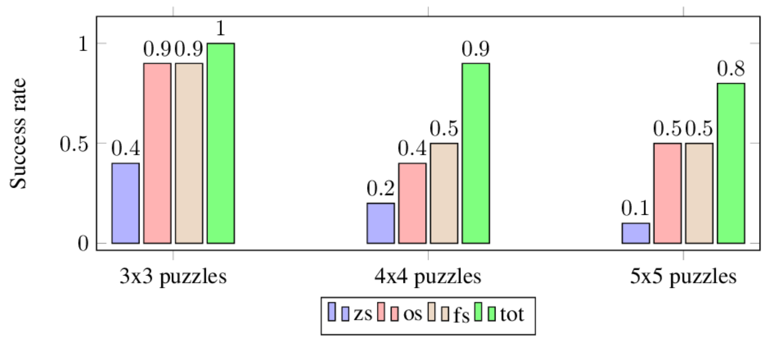


Figure 5.3.1: Experimental results comparing the success rate of different LLM-based Sudoku puzzle solvers across three sets of benchmarks [74].

Rubik's Cube and **Maze Solvers** have also been used to evaluate LLM performance in spatial reasoning tasks. Noever et al. (2021) [87] assessed GPT-2's ability to solve spatial puzzles using a dataset of over 2,400 Rubik's Cube samples and 10,000 maze-solving examples. Despite limitations in token constraints and fine-tuning duration, GPT-2 demonstrated potential by successfully solving the Rubik's Cube in 1 out of 7 attempts, though many outputs were valid yet incorrect solutions. These results suggest that while LLMs have the capacity for spatial reasoning, more advanced approaches are necessary for achieving consistent accuracy. In another study, Ding et al. (2023) [25] tested multiple methods, including CoT, SC, ToT, and

GoT, on a 2×2×2 Rubik's Cube using GPT-3.5 and GPT-4. Their results showed that the Everything-of-Thought (XoT) method, significantly outperformed other approaches, achieving a 77.6% success rate. This demonstrates how advanced thought structures and iterative reasoning can enhance LLM performance in deterministic puzzles with spatial complexity.

LLM versatility was further tested on the **8-Puzzle** and the **Game of 24**, both of which test numerical and spatial reasoning in deterministic environments [25]. In the 8-Puzzle, where solvers must rearrange tiles to achieve a goal configuration, the XoT method achieved an impressive 93.2% accuracy across 419 puzzles, showcasing the power of structured reasoning and self-revision over traditional few-shot prompting or CoT methods. Similarly, in the Game of 24—a numerical puzzle where players must manipulate four numbers with mathematical operations to reach a target value of 24—XoT outperformed other methods, further solidifying its effectiveness in deterministic puzzle-solving contexts.

Crosswords provide another challenging domain for LLMs, particularly in understanding and decoding wordplay and cryptic clues. Research works fine-tuned T5 models [101] on extensive datasets of individual cryptic crossword clues [104, 28]. Their findings highlighted T5's advantage over traditional non-neural baselines, although challenges remained, especially with quick clues and those requiring precise answer lengths. A study compared BART [64] and T5 models, revealing suboptimal performance with accuracy below 30% for clue-answer tasks, though retrieval-augmented generation transformers showed promising improvements [59]. Yao et al. (2023) [134] further applied 5-shot prompting and ToT to GPT-4 on crossword puzzles, achieving a notable improvement, solving 4 out of 20 puzzles and obtaining a 60% word-level success rate. These results underscore the difficulties LLMs face in tasks requiring linguistic creativity and inference, while also showcasing the potential of advanced thought structures to boost performance.

In the domain of **Chess puzzles**, Feng et al. (2023) [30] fine-tuned two specialized models—"ChessGPT" and "ChessCLIP"—using a vast dataset from Lichess comprising 3.2 million puzzles[3]. Each puzzle was annotated with details like the puzzle's rating, theme, and solution, allowing the models to learn patterns that correlate with successful moves. Fine-tuning on this dataset significantly improved the models' ability to solve chess puzzles, demonstrating the effectiveness of task-specific data in enhancing LLM performance in deterministic, rule-based games.

Finally, Kazemi et al. (2023) [51] introduced **BoardgameQA**, a dataset specifically designed for testing models' abilities to navigate rule-based environments with contradictory facts and complex rules. The dataset includes multi-choice questions requiring free-text answers, offering a comprehensive test of LLMs' ability to reason within structured game contexts. In their evaluation, fine-tuning BERT-large and T5-XXL models with proofs yielded the best performance, outperforming CoT-based few-shot prompting on PaLM. The study also noted a decrease in accuracy when extra or conflicting information was introduced, highlighting the challenge of handling noisy data in puzzle-solving contexts.

Stochastic Puzzles

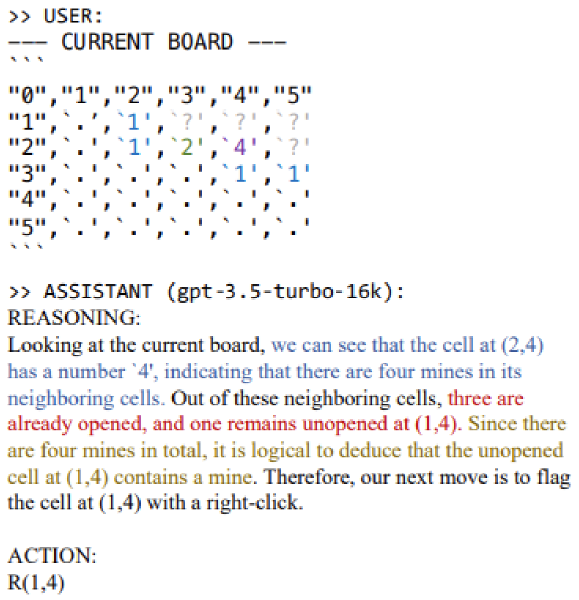
Stochastic puzzles introduce a layer of complexity through randomness or hidden information, making them distinct from deterministic puzzles. In these puzzles, the same player action can lead to varying outcomes depending on unknown factors or probabilistic elements. This unpredictability presents significant challenges for LLMs, requiring them to reason not only about the immediate game state but also to plan for multiple potential future states.

The **BoardgameQA** benchmark [51] serves as a prime example of stochastic puzzle challenges. This dataset besides the puzzles presented before, also incorporates puzzles where essential information is missing, compelling the model to reason under uncertainty. The authors demonstrated that as the amount of missing information increases, the accuracy of fine-tuned models significantly decreases. However, this decline does not uniformly affect all methods; prompt-tuned and few-shot learning methods appear less impacted by the increasing complexity, likely due to the larger, more sophisticated models applied during evaluation. This discrepancy suggests that different reasoning strategies might be necessary depending on the model's size and the specific challenges posed by missing or uncertain data in stochastic puzzles.

One of the most iconic stochastic puzzles is **Minesweeper**, a game characterized by hidden information

[3]https://lichess.org/

where players must deduce the locations of hidden mines based on numerical clues. The unpredictability of mine locations requires the model to combine spatial reasoning with probabilistic inference. Li et al. (2023) [66] explored how various LLMs perform on Minesweeper by evaluating two different input formats: table representations, where the grid is visually represented, and coordinate-based representations, which encode the game's clues as positions on a coordinate plane. GPT-3.5, despite demonstrating some initial understanding of the game mechanics, showed limited improvement when applying few-shot prompting techniques. In contrast, GPT-4 exhibited a stronger ability to identify mines but still struggled to complete entire boards, revealing gaps in strategic thinking and long-term planning. Interestingly, the experiments suggested that the coordinate representation format was more effective in helping the LLM comprehend Minesweeper's spatial reasoning challenges than the table format.



~~~
>> USER:
--- CURRENT BOARD ---
```

"0","1","2","3","4","5"
"1",`.',`1',`?',`?',`?'
"2",`.',`1',`2',`4',`?'
"3",`.',`.',`.',`1',`1'
"4",`.',`.',`.',`.',`.'
"5",`.',`.',`.',`.',`.'
```

>> ASSISTANT (gpt-3.5-turbo-16k):
REASONING:
Looking at the current board, we can see that the cell at (2,4)
has a number `4', indicating that there are four mines in its
neighboring cells. Out of these neighboring cells, three are
already opened, and one remains unopened at (1,4). Since there
are four mines in total, it is logical to deduce that the unopened
cell at (1,4) contains a mine. Therefore, our next move is to flag
the cell at (1,4) with a right-click.

ACTION:
R(1,4)
~~~

Figure 5.3.2: A case study of a "valid" action and its corresponding reasoning generated by GPT-3.5-16k for solving Minesweeper. Blue indicates logical reasoning; red and golden are illogical ones [66].

Card games, especially Poker, present another key category of stochastic puzzles where hidden information plays a central role. Poker requires players to infer the hidden cards of opponents, calculate probabilities, and make strategic decisions under uncertainty. In simplified Poker variants, these elements are even more pronounced, as players must assess not only their own hand but also the possible intentions of others based on partial information. Gupta et al. (2023) [42] investigated ChatGPT's and GPT-4's abilities in the pre-flop round of Poker, observing that while both models grasped basic Poker strategies, neither reached Game Theory Optimal (GTO) play. GPT-4 was more aggressive in its gameplay, while ChatGPT tended to adopt a more conservative approach, likely due to differences in their training data and reasoning patterns. In more advanced experiments, Huang et al. (2024) [44] trained the OPT-1.3B model using Reinforcement Learning (RL) to play Poker, achieving superior win rates and efficiency across all phases of the game. This study highlights the potential of combining RL with LLMs to enhance decision-making in stochastic environments where incomplete information requires probabilistic reasoning and strategic play. Moreover, a recent study revealed that an agent leveraging GPT-4 achieved remarkable success in various imperfect information card games, further demonstrating the adaptability of LLMs in strategic, uncertainty-driven scenarios [41].

In addition to card games, **social deduction games** such as Werewolf and Avalon also fall under the category of stochastic puzzles. These games blend logical reasoning with complex social dynamics, requiring players to deduce the hidden roles and intentions of others based on incomplete information and behavioral cues. Werewolf, for example, challenges players to identify the "werewolves" within a group, while the "villagers" attempt to survive. Xu et al. (2023) [131] proposed a framework where LLMs are applied to Werewolf without any fine-tuning, utilizing historical in-game interactions to inform their strategic decisions. This framework demonstrated that LLMs could successfully navigate the game's social complexities, although not perfectly, indicating their potential for understanding social reasoning. Similarly, frameworks for Avalon, highlighted how LLMs can handle scenarios that demand both logical deduction and social manipulation [121, 60]. These studies underscore the models' proficiency in managing the interplay between logical reasoning and social interaction, which is essential for success in social deduction games.

Stochastic puzzles represent a rich testing ground for evaluating LLMs' abilities to handle uncertainty and make strategic decisions under imperfect conditions. While the models have shown promise in various applications, from Minesweeper to Poker and social deduction games, their performance often depends on the specific methods employed, such as fine-tuning, prompting, or the integration of external engines. These results suggest that LLMs are evolving in their ability to process and navigate the complexities inherent in stochastic puzzles, although there remains significant room for improvement, particularly in terms of strategic depth and probabilistic reasoning.

5.3.2 Rule-less Puzzles

This subsection delves into the diverse datasets associated with rule-less puzzles, a category encompassing riddles, programming puzzles, and commonsense reasoning challenges. Rule-less puzzles differ from their rule-based counterparts in that they typically lack formal game mechanics or predefined moves. Instead, they emphasize real-world knowledge, inferential reasoning, and lateral thinking. In the context of puzzle-solving with LLMs, these datasets primarily serve as benchmarks to evaluate the model's ability to navigate ambiguous scenarios, draw inferences, and connect disparate pieces of information to arrive at solutions.

Notably, the rule-less puzzle category is distinct from code generation tasks, which involve generating executable code based on specific problem statements. While programming puzzles are included, they are discussed in the traditional sense of problem-solving through code logic, not the broader task of code generation. Similarly, we exclude datasets specifically designed for open-ended code generation, which, while related, pose different challenges from the reasoning tasks that define rule-less puzzles.

A majority of the rule-less puzzle datasets are structured in a **multiple-choice question-answering (QA)** format. This approach allows for standardized evaluation of LLM performance, as it offers clear metrics for assessing correctness based on predefined answers. In these settings, the models must reason through incomplete or ambiguous information, often relying on commonsense knowledge or abstract problem-solving strategies. The multiple-choice format presents an efficient mechanism for evaluating LLMs' inferential reasoning by providing a controlled environment in which different reasoning paths can be tested.

However, not all rule-less puzzles follow this format. Some benchmarks employ more open-ended tasks, which demand more intricate reasoning and creative problem-solving from the models. These benchmarks introduce a greater degree of complexity by allowing for multiple valid answers or by challenging the models to explain their reasoning, rather than simply providing a correct output. Such variations push the boundaries of LLM capabilities, offering insight into how well these models perform under less structured, more flexible problem-solving scenarios. Where applicable, these benchmarks are highlighted in order to provide a more comprehensive view of how different datasets assess LLMs' ability to handle rule-less puzzles.

Riddles

Riddles, a classic form of word puzzle, challenge LLMs to draw upon linguistic nuances, abstract thinking, and lateral reasoning. Their ambiguous nature, reliance on metaphor, and clever wordplay test a model's ability to engage in creative and non-linear problem-solving. Datasets in this domain provide varied formats and task designs, with a mix of multiple-choice questions, free-text responses, and language generation challenges.

RiddleSense [67] is one of the most prominent datasets for riddles, comprising 5.7K vertical thinking puzzles that emphasize abstract reasoning. The dataset primarily tests models such as BERT, RoBERTa, ALBERT, and text-to-text QA models like UnifiedQA [52] and T5. These LLMs are evaluated based on their ability to infer answers through vertical reasoning, a process that demands logic-based pattern recognition. Larger models such as UnifiedQA (based on T5-3B) demonstrate superior performance, but continue to struggle with complex elements such as metaphors and counterfactual situations. RiddleSense's evaluation underscores the fact that while larger models benefit from scale, they still exhibit notable weaknesses in lateral or creative reasoning, areas where human cognition excels.

BrainTeaser [50] takes a step further by introducing 1119 lateral thinking puzzles, a domain that further tests models' capacity for divergent thinking. In contrast to vertical reasoning puzzles, lateral thinking challenges involve scenarios that require non-obvious connections and out-of-the-box thinking. The dataset contrasts the performance of instruction-based models like ChatGPT, T0, and FlanT5 [19] with that of commonsense-based models such as RoBERTa variants and CAR [122]. ChatGPT, despite its general-purpose design, excels in sentence-based and word-based puzzles, demonstrating its strengths in lateral thinking. However, the dataset reveals that even state-of-the-art LLMs tend to fall into traps of memorization and shallow commonsense reasoning, emphasizing the need for more advanced methods of inference and reasoning. Through fine-tuning Mistral-7b [49], Panagiotopoulos et al. (2023) [91] have achieved to outperform the best neural baseline (ChatGPT) by more than 20% and 30% for the sentence puzzles and the word puzzles respectively.

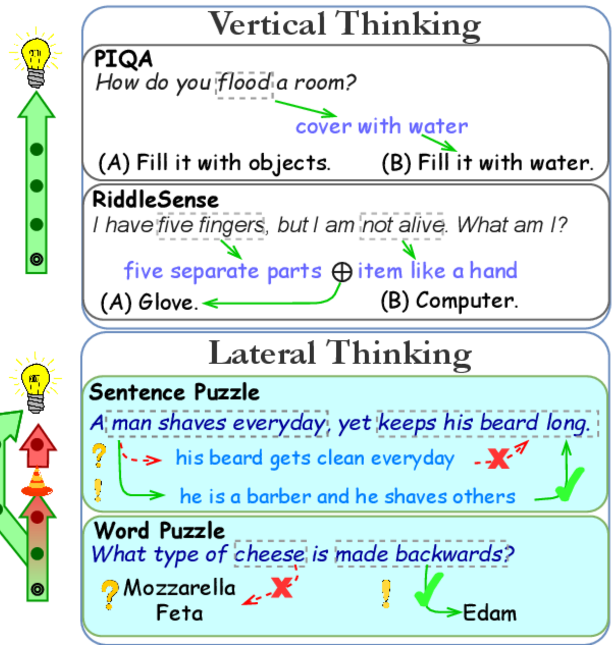

Vertical Thinking
PIQA
How do you flood a room?
cover with water
(A) Fill it with objects.
(B) Fill it with water.
RiddleSense
I have five fingers, but I am not alive. What am I?
five separate parts
item like a hand
(A) Glove.
(B) Computer.
Lateral Thinking
Sentence Puzzle
A man shaves everyday, yet keeps his beard long.
his beard gets clean everyday
he is a barber and he shaves others
Word Puzzle
What type of cheese is made backwards?
Mozzarella
Feta
Edam

Figure 5.3.3: Contrasting Vertical Thinking tasks (PIQA (Bisk et al., 2020 [9]) and RiddleSense (Lin et al., 2021 [67])) to BRAINTEASER, a lateral thinking task.

BiRdQA [139] adds another layer of complexity by introducing multilingual riddles, offering puzzles in both English and Chinese. This dataset evaluates the performance of both monolingual models (BERT, RoBERTa) and multilingual ones (mBERT, XLM-R [21]). It also tests the efficacy of using brief riddle introductions and hints. A key finding from BiRdQA is the significant performance gap between LLMs and human-level understanding, particularly in multilingual contexts. Monolingual models consistently outperform their multilingual counterparts, and the effectiveness of hints varies based on the language—such aids prove helpful for English riddles but less so for Chinese riddles, hinting at potential linguistic or cultural biases that affect model comprehension.

On the Chinese front, **CC-Riddle** [128] presents a vast dataset of 27K Chinese character riddles in multiple-choice, generative, and retrieval-based formats. This dataset challenges LLMs with the intricate task of understanding Chinese characters, which introduces a layer of semantic complexity. Evaluations reveal that models like GPT-3 and GPT-4 struggle to effectively comprehend and process these riddles, often demonstrating significant misunderstandings. This indicates that character-based riddles, particularly in languages like Chinese, introduce additional challenges for LLMs, highlighting the need for more targeted pre-training or fine-tuning to address language-specific nuances.

Complementing these, **PUZZLEQA** [142] offers 558 word puzzles, testing models in both multiple-choice and free-text response formats. While larger models like GPT-3/3.5 show higher accuracy in multiple-choice settings, free-response puzzles prove more challenging, even for models fine-tuned for QA tasks. Interestingly, prompting techniques like CoT combined with summarization do not significantly improve performance on free-response puzzles, underscoring the complexity of free-form generation tasks in puzzle-solving scenarios.

Finally, **MARB** [116] encompasses a variety of riddle tasks, pushing the boundaries of model reasoning by integrating several prompting strategies, including zero-shot, CoT, and IEP. Tested on models like GPT-4 and PaLM2-540B [2], the combination of IEP and CoT emerges as the most effective approach, yielding the highest success rates across diverse riddle types. The dataset also covers commonsense puzzles, further demonstrating that integrating multiple reasoning techniques provides substantial benefits when dealing with ambiguous or open-ended problems.

In summary, the landscape of riddle datasets offers rich opportunities to probe the boundaries of LLM reasoning. The diversity of datasets, from monolingual to multilingual riddles, and from structured to free-response formats, highlights the areas where LLMs show promise—and the gaps where human cognition still holds an edge, especially in the realms of creativity, abstraction, and metaphorical thinking.

Programming Puzzles

Programming puzzles present a unique category of problem-solving tasks, requiring models to engage with code semantics, logic, and algorithmic reasoning. These challenges test an LLM's ability to understand programming languages, predict outcomes of code execution, and even generate syntactically and semantically correct code. Unlike riddles, which often rely on abstract thinking or commonsense reasoning, programming puzzles demand precise, step-by-step reasoning capabilities, often requiring the model to "run" or simulate parts of the code mentally.

One of the leading datasets in this domain is **P3 (Python Programming Puzzles)** [107]. P3 offers a broad spectrum of challenges, ranging from basic tasks like string manipulations and arithmetic operations to more complex algorithmic puzzles like Tower of Hanoi. These puzzles are designed with the goal of finding an input that makes the Python program f return "True". This setup presents a reverse engineering challenge where the model must deduce the correct input by reasoning about the underlying logic of the provided function. In the evaluation of these tasks, models such as GPT-3 and Codex [15] have been employed, using various prompting techniques to guide their responses. The pass@k metric, a key measure of performance, indicates the model's ability to solve a puzzle within k attempts, offering insights into both efficiency and accuracy.

A noteworthy finding from the evaluations is the correlation between puzzle difficulty for models and human solvers alike. The more complex the puzzle, the more attempts it took for both humans and LLMs to solve it. In particular, descriptive prompts that clearly outline the problem's parameters and constraints significantly enhance the performance of models like Codex. This highlights the critical role that prompt design plays in guiding LLMs through complex tasks, particularly those requiring algorithmic thinking. Additionally, models proficient in code completion solved more puzzles with fewer tries, showcasing the importance of specialized capabilities in handling programming challenges. These results illustrate that while LLMs are proficient in tasks such as code generation and completion, their success heavily depends on how the problem is framed within the prompt.

Further contributions to the study of programming puzzles come from Savelka et al. (2023) [106], who introduce a dataset of 530 code snippets derived from programming courses. This dataset, unlike P3, presents puzzles in a multiple-choice format. The dataset is unique in that it distinguishes between questions with and without accompanying code snippets, providing a nuanced perspective on how LLMs handle different types of programming problems. The puzzles are categorized into six distinct types, including true/false questions, output prediction, and program correction. This structured categorization enables a deeper evaluation of the specific types of reasoning that LLMs apply to different problem types.

When evaluated on the **Programming Snippets Dataset**, GPT models demonstrated varying accuracy rates across the different question types. Code snippets added a layer of complexity to the puzzles, which led to a decrease in overall accuracy compared to more straightforward questions. However, the models performed notably better on completion-oriented tasks, such as output prediction, where the question format clearly delineated the expected outcome. These findings suggest that the complexity of programming puzzles—and the effectiveness of LLMs in solving them—can depend significantly on the format and structure of the question itself. A clearly defined question, particularly one where the model's task is to predict an output or complete a given code snippet, allows the model to leverage its pre-trained knowledge more effectively.

While both P3 and the Programming Snippets Dataset focus on programming puzzles, they do so in distinct ways. P3 emphasizes the reverse-engineering process by tasking the model with finding the correct inputs to make a function return a desired result, whereas the Programming Snippets Dataset uses a multiple-choice format to test comprehension of code snippets and programming logic. Despite these differences, both datasets yield valuable insights into LLM performance: descriptive and well-structured prompts aid in problem-solving, and the question format heavily influences model success.

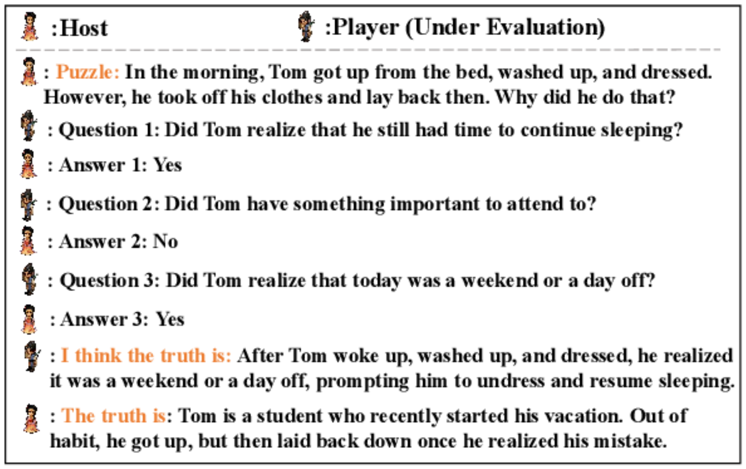
Commonsense Reasoning Puzzles

Commonsense reasoning puzzles present a unique challenge to LLMs, as they require models to go beyond simple logical deduction and engage in inferential thinking that draws on real-world knowledge and context. These puzzles often test a model's ability to fill in gaps, interpret implicit information, and make plausible conclusions based on everyday situations. In this subsection, we examine datasets specifically designed to test LLMs' commonsense reasoning abilities in puzzle contexts.

True Detective [23] introduces a set of detective puzzles embedded in long-form narratives. These puzzles require LLMs to read through the story, analyze clues, and draw logical conclusions. True Detective is notable for its complexity, as all the information required to solve the puzzle is provided within the story, yet the challenge lies in connecting disparate clues and forming accurate inferences. LLMs such as GPT-3.5 and GPT-4 are evaluated on these tasks, using methods like CoT prompting and a specialized variant known as Golden-CoT. Golden-CoT provides the model with the reasoning process behind the correct answer, guiding it towards understanding the logical steps involved before extracting the final answer. Despite this guidance, models often struggle to reach the correct solution, with Vanilla and CoT approaches performing close to random on these tasks. GPT-4, however, performs significantly better with Golden-CoT, matching human solver performance, while GPT-3.5 achieves a solve rate of only 63%. This disparity underscores the difficulty of these puzzles, even for advanced LLMs, and highlights the critical role that guided reasoning plays in enhancing performance on detective-style problems.

DetectBench [40] evaluates informal reasoning in more real-life contexts, providing 1,200 questions that challenge models to apply commonsense knowledge to solve real-world puzzles. It tests various prompting methods, such as the use of hints, different CoT approaches, and a detective thinking technique across models like GPT-3.5, GPT-4, GLM-4 and Llama2. Hints are shown to play a crucial role in aiding both model and human performance, often significantly improving outcomes, particularly for larger models. Detective thinking—a strategy that encourages the model to process information more methodically—proves highly effective for guiding reasoning, especially in complex scenarios. These findings reveal that even with powerful models, prompting techniques and additional cues, such as hints, are essential for navigating the intricacies of commonsense reasoning puzzles.

LatEval [47] introduces a conversational puzzle-solving format, focusing on interactive reasoning where players ask yes/no questions before arriving at a final solution. The dataset includes English and Chinese stories, testing LLMs like GPT-3.5, GPT-4, and various chat-based models on their ability to ask relevant, consistent questions that align with the puzzle's constraints. Unlike typical multiple-choice or free-text puzzles, LatEval requires active engagement, where the quality of questions and the consistency of answers are critical factors in success. Larger models generally perform better in maintaining consistency and answering truthfully, yet performance in generating relevant, probing questions remains mixed. GPT-4 demonstrates the highest level of answer consistency, although there is still considerable room for improvement in interactive commonsense reasoning.


:Host :Player (Under Evaluation)

: Puzzle: In the morning, Tom got up from the bed, washed up, and dressed. However, he took off his clothes and lay back then. Why did he do that?

: Question 1: Did Tom realize that he still had time to continue sleeping?

: Answer 1: Yes

: Question 2: Did Tom have something important to attend to?

: Answer 2: No

: Question 3: Did Tom realize that today was a weekend or a day off?

: Answer 3: Yes

: I think the truth is: After Tom woke up, washed up, and dressed, he realized it was a weekend or a day off, prompting him to undress and resume sleeping.

: The truth is: Tom is a student who recently started his vacation. Out of habit, he got up, but then laid back down once he realized his mistake.

Figure 5.3.4: An example of a Lateral Thinking Puzzle in the LatEval benchmark [47].

Another intriguing resource is **PuzzTe** [112], which offers a collection of puzzles centered on comparative reasoning, including knights and knaves problems and zebra puzzles. Though not yet applied to LLMs, the dataset represents a potential benchmark for testing models on structured reasoning tasks. Solutions are generated using tools such as the Mace4 model finder and Prover9 theorem prover[4], offering a more symbolic approach to puzzle solving. This dataset holds promise for future evaluations of LLM performance in handling formal logic and comparative puzzles, providing a potential bridge between symbolic reasoning and language-based inference.

Together, these datasets illustrate the diverse challenges posed by commonsense reasoning puzzles. Detective-style tasks, as seen in True Detective and DetectBench, push LLMs to make inferences based on real-world knowledge and hidden information, often requiring advanced prompting methods like Golden-CoT to achieve reasonable accuracy. Meanwhile, interactive puzzle-solving datasets like LatEval highlight the complexity of maintaining consistency and relevance in conversational reasoning, a critical aspect of commonsense inference. Larger models, such as GPT-4, generally outperform smaller counterparts, but there remains a significant gap between the performance of LLMs and human reasoning, particularly in tasks that require nuanced, real-world understanding. Additional methods like providing hints or step-by-step reasoning prompts can help bridge this gap, but even with these aids, commonsense reasoning puzzles remain a formidable challenge for LLMs.

It is important to note that this work focuses on puzzle-oriented benchmarks specifically. Broader commonsense reasoning datasets, such as CommonsenseQA, PIQA [9], or StrategyQA [36], which test general commonsense knowledge outside the puzzle-solving context, are not included in this analysis.

5.4 Literature Review Discussion

5.4.1 Applied Methods and Dataset Gaps

Through the exploration of puzzle-solving datasets and benchmarks, various methodological trends have emerged, particularly in the use of prompting techniques, fine-tuning and puzzle translation. Across our puzzle taxonomy, methods like few-shot prompting, CoT, and the use of supplementary information such as introductions or hints are common in both rule-based and rule-less puzzles. However, there are notable differences in the diversity of approaches depending on the puzzle type. For instance, rule-based deterministic puzzles, such as Sudoku and Rubik's Cube, and rule-less commonsense puzzles exhibit a broad variety of methods. In contrast, rule-based stochastic and rule-less programming puzzles show fewer studies and methods applied, reflecting the research gap in these more complex domains.

[4]https://www.cs.unm.edu/ mccune/prover9/

Our analysis highlights a wealth of datasets available for rule-based deterministic puzzles (e.g., Sudoku, 8-puzzle, crosswords) and rule-less riddles, which are frequently used to assess LLMs' cognitive reasoning. This demonstrates the extensive research interest and resource availability in these domains. However, as illustrated in the previous sections, there remains a significant scarcity of datasets for rule-based stochastic puzzles, such as Minesweeper and Poker, as well as rule-less programming puzzles. These gaps provide promising opportunities for further research, particularly in creating more diverse benchmarks that challenge the problem-solving capabilities of LLMs. The limited availability of benchmarks for stochastic puzzles also underscores the need for more datasets that incorporate elements of randomness and hidden information, which are essential for advancing reasoning techniques under uncertainty.

Additionally, while neuro-symbolic approaches that translate natural language into formal representations have proven valuable, they remain underutilized in puzzle-solving benchmarks. Most studies focus on logic puzzles or specific code-based challenges but have yet to explore their full potential across other puzzle types. For example, methods like PoT, PAL and Faithful-CoT, while primarily tested on mathematical or logical reasoning, hold potential for broader applications in puzzle-solving tasks, particularly where structured problem representation is essential. This area remains relatively unexplored, suggesting a new direction for further investigation.

5.4.2 Performance Analysis

Different categories of puzzles exhibit varied performance results for LLMs based on the methods applied:

- *Rule-based/Deterministic Puzzles:* Methods like ToT, GoT and XoT (§5.2) have demonstrated their efficacy in enhancing model reasoning by structuring thought processes in a hierarchical or graph-based manner. This structural complexity significantly improves performance, especially in puzzles like the Game of 24 or 8-puzzle, where the model benefits from multiple reasoning paths. However, datasets like BoardgameQA and crossword puzzles continue to challenge LLMs, reflecting ongoing limitations in strategic reasoning and complex deduction even within well-structured environments.
- *Rule-based/Stochastic Puzzles:* Fine-tuning has been applied successfully to certain rule-based stochastic puzzles, allowing models to understand basic rules and predict outcomes under simpler scenarios. However, in more intricate puzzles such as Minesweeper, LLMs often falter, struggling with probabilistic reasoning, multi-step planning, and risk management in environments where hidden information or randomness plays a critical role.
- *Rule-less/Riddles & Commonsense Puzzles:* There remains a substantial performance gap between LLMs and human solvers in tasks involving riddles or commonsense reasoning. While techniques like CoT improve accuracy, they are still far from achieving human-level understanding and inferencing. The complexity of linguistic subtleties, metaphors, and abstract reasoning continues to hinder model performance, as shown in datasets like RiddleSense and BrainTeaser, where LLMs often fail to generalize effectively.
- *Rule-less/Programming Puzzles:* Programming puzzles represent a particularly challenging category, even for large LLMs such as Codex and GPT-4. Results from datasets like P3 (Python Programming Puzzles) and Programming Snippets suggest that LLMs face similar difficulties to humans in complex problem-solving tasks [107], while tasks involving code analysis and reasoning in multiple-choice formats prove particularly tough [106].

Furthermore, the format of questions significantly affects puzzle-solving effectiveness. Multiple-choice setups simplify tasks for LLMs by narrowing the solution search space, while free-text formats increase the difficulty level.

5.4.3 Puzzle Generation

The generation of puzzles by LLMs is a relatively underexplored area. Current research has primarily focused on puzzle-solving rather than puzzle creation. The few studies in puzzle generation report mixed outcomes. For example, GPT-3.5's attempts to generate riddles with corresponding solutions often produce unsatisfactory results [142], highlighting the need for further refinement in this area. In contrast, recent works such as ACES (Autotelic Creation of Programming Puzzles) demonstrate promising advancements in generating diverse programming puzzles using semantic descriptors generated by LLMs [97]. Lastly, cross-lingual puzzle generation has also been explored, with models generating cryptic crossword puzzles in multiple languages [145, 137, 138]. However, these efforts are still in their early stages, and there is ample room for improvement in both the quality and diversity of generated puzzles.

Chapter 6

Experiments

The goal of this chapter is to empirically evaluate the reasoning and puzzle-solving abilities of Large Language Models (LLMs) across a variety of datasets and methods. Specifically, we examine the performance of smaller models, such as Llama2-7B, Llama3-8B, Llama3.1-8B, Llama3.1-70B, and Mistral-7B, (all of the models are instruct-tuned) on both mathematical reasoning and puzzle-solving tasks. Previous research has demonstrated the effectiveness of advanced prompting techniques like Chain-of-Thought (CoT), neurosymbolic methods, and least-to-most (LtM) prompting on larger models (e.g. Codex models or GPT-4). However, the application of these techniques to smaller models remains underexplored, especially in domains that require multi-step logical reasoning and problem-solving under constrained computational resources.

In this chapter, we focus on four **datasets**: two mathematical reasoning datasets—*GSM8K* and *SVAMP*—and two puzzle reasoning datasets, *RiddleSense* and *Game-of-24*. We apply several prompting techniques (IO prompting, CoT, LtM and Faithful-CoT / PoT) under various settings, including zero-shot, few-shot, and self-consistency prompting. These techniques, particularly neurosymbolic ones, have shown promising results with larger models, but their effectiveness with smaller, more resource-efficient models is yet to be fully understood. Moreover, neurosymbolic methods—where problems are converted into code for external reasoning engines—have rarely been tested for puzzle-solving as discussed in the previous chapter, offering an innovative direction explored in this study.

The experiments are structured to assess the following:

- The performance of smaller LLMs in mathematical and puzzle-solving tasks, focusing on their ability to handle complex reasoning in both structured (rule-based) and unstructured (rule-less) environments.
- The effectiveness of neurosymbolic puzzle translation, specifically applied to puzzles, and how it compares to more common prompting methods.
- Whether advanced prompting methods like Faithful-CoT and PoT can enable emergent reasoning abilities in smaller models, which have not been explicitly designed for such tasks.

Through these experiments, we aim to provide insights into the capabilities and limitations of smaller models when faced with tasks typically dominated by much larger LLMs. Additionally, the results will offer a deeper understanding of how innovative prompting strategies can enhance the problem-solving capabilities of LLMs in puzzle contexts.

Contents

6.1 Preliminaries

6.1.1 Datasets

In this section, we present the datasets used in our experiments, which include two mathematical reasoning datasets—GSM8K and SVAMP—and two puzzle-solving dataset, RiddleSense and the Game-of-24. Each dataset offers unique challenges that test the ability of LLMs to perform mathematical reasoning, as well as puzzle-solving tasks that demand logical inference and pattern recognition. All datasets, except Game-of-24, fall under the rule-less category, as they do not involve specific rules that the model should follow to find the correct answer. For the Game-of-24, on the other hand, the models should follow standard steps in order to find the solution.

While the mathematical reasoning datasets have been used previously to evaluate the Faithful-CoT and PoT methods, those evaluations involved larger models. In our experiments, we aim to test the effectiveness of neurosymbolic methods, such as Faithful-CoT, on the Game-of-24 puzzle—an unexplored task in this context.

GSM8K (Grade School Math 8K)

GSM8K [20] is a dataset specifically created to evaluate the mathematical reasoning abilities of large language models. It includes 8.5K math problems, split into 7.5K training and 1K test problems. Our experiments are conducted on the test set with the *1K problems*. Each problem typically requires 2 to 8 logical steps to arrive at the solution, with the majority focusing on basic arithmetic operations like addition, subtraction, multiplication, and division. These problems are designed to be solvable by a middle school student, making GSM8K an appropriate challenge for evaluating the problem-solving capabilities of LLMs.

Key Features:

- **Human-Curated Problems:** GSM8K stands out for its high-quality, human-written problems, which are carefully crafted to ensure accuracy. This approach contrasts with other datasets that often rely on automatic scraping, which can introduce noise or errors. Extensive quality control measures are used to verify the correctness of the problems, leading to a dataset with a very low error rate (less than 2%).
- **Diversity of Problems:** One of the key design principles of GSM8K is to provide a wide range of problems that are not just variations of a common template. Each problem is relatively unique, both in structure and context, ensuring that the dataset covers a broad spectrum of mathematical scenarios. This makes the test set an important benchmark for understanding how well models can generalize across different problem formats.
- **Moderate Difficulty:** GSM8K is carefully balanced to challenge even advanced language models while still being within reach for large models with sufficient reasoning capabilities. The problems require basic arithmetic and early algebra, but they do not typically demand higher-level concepts like advanced calculus or formal variable manipulation. This balance of difficulty makes GSM8K particularly useful for assessing how different models scale their abilities as they process increasingly complex problems.
- **Natural Language Problem Solving:** A distinctive aspect of GSM8K is that solutions are written in natural language. This design choice reflects the importance of LLMs not only performing calculations but also explaining their reasoning process in a way that mimics human thinking. Problem writers are encouraged to provide detailed, step-by-step explanations in their own linguistic styles, which introduces a variety of expressions and further challenges LLMs to adapt to different ways of presenting logical reasoning.

SVAMP (Simple Variations on Arithmetic Math word Problems)

SVAMP [94] is a benchmark dataset designed to evaluate the robustness of models in solving elementary-level math word problems (MWPs). The dataset contains *1,000 arithmetic problems*, specifically focusing on one-unknown problems that are meant to challenge models across different reasoning aspects. SVAMP problems involve simple expressions that can be solved using no more than two operators (e.g., addition, subtraction, multiplication, division), making them accessible for early grade-level students.

Key Features:

- **Human-Curated Quality Control:** The dataset was meticulously curated by human annotators familiar with math word problems. After the initial set of 1,098 problems was created, only 1,000 were included in the final dataset after excluding problems that might be unfairly difficult due to complexity beyond the targeted elementary level. To ensure consistency and accuracy, the dataset underwent additional review by volunteers unfamiliar with the task, focusing on grammatical and logical correctness.
- **Moderate Difficulty with High Challenge:** While the individual problems in SVAMP might seem straightforward to humans, requiring only basic arithmetic, they present substantial challenges for models. The dataset includes only problems solvable with two arithmetic operators at most, yet the diversity in problem structure makes it more difficult for models than other similar datasets like ASDiv-A [82] and MAWPS [54]. This balance between simplicity in arithmetic operations and complexity in problem structure ensusres that models cannot rely on shortcuts or memorized patterns to solve the problems.

RiddleSense

RiddleSense [67] is a dataset designed to evaluate the capabilities of language models in solving riddles, which are a type of rule-less puzzle that require flexible thinking, language understanding, and commonsense reasoning. The dataset challenges models to interpret and solve problems that involve clever wordplay, metaphors, and indirect clues.

Key Features:

- **Dataset Composition:** Comprises 5.7K vertical thinking, multiple-choice riddles curated to challenge language models on tasks that require inference and abstract reasoning. However, in our experiments we use only 3k random riddles from the dataset. Each riddle is presented in natural language, demanding models to leverage both linguistic skills and world knowledge to arrive at the correct answer.
- **Problem Complexity:** The riddles range from straightforward to complex, involving wordplay, metaphorical phrasing, and lateral thinking. For instance, a typical riddle might be: "What gets wetter the more it dries?" with the answer being "a towel". Models need to understand the dual meaning and metaphorical context.
- **Reasoning Abilities:** RiddleSense primarily tests models' abilities to think beyond explicit cues, requiring them to infer hidden connections between clues. This makes it an ideal dataset for assessing a model's performance on tasks that go beyond pattern recognition, and it emphasizes the need for broad, flexible thinking.

Game of 24

The Game of 24 dataset [134], used to evaluate mathematical reasoning, challenges models to manipulate four numbers using basic arithmetic operations (addition, subtraction, multiplication, division) to produce the result 24. This task requires the model to find a valid combination of operations that can be applied to the input numbers in various possible orders to achieve the target number. For example, given the numbers "4, 9, 10, 13" a correct solution might be expressed as "(10 - 4) * (13 - 9) = 24."

Game of 24 falls into the rule-based deterministic category, as the solution process is strictly governed by mathematical rules. Each problem has a clearly defined goal (reaching 24) and requires logical reasoning and arithmetic operations applied in a specific order to solve.

Key Features:

- **Dataset Structure:** The dataset consists of 1,362 entries in total, with 1,225 problems used for training and 137 problems reserved for testing. These problems are ranked by human solving time, which indirectly provides a measure of difficulty. The testing set of *137 entries*, which covers a range of difficulty levels, forms the basis for evaluating the model's performance in this study. Each problem in the dataset presents four numbers ranging from 1 to 13, which must be used exactly once in the solution. The objective is to apply a combination of arithmetic operations on these numbers to achieve the final result of 24. While some problems may have multiple valid solutions, the model is expected to find at least one correct answer that follows the rules of arithmetic.

- **Challenges and Importance:** The Game of 24 dataset provides a rigorous test of multi-step reasoning and arithmetic problem-solving for LLMs. Unlike simpler math problems, this task requires strategic exploration of possible combinations of numbers and operations. Moreover, the inclusion of varying difficulty levels, as measured by human solving time, ensures that models are tested not only on straightforward calculations but also on more complex reasoning tasks that involve trial-and-error thinking. The dataset's structure, particularly its ranking by difficulty, allows for nuanced evaluations of a model's ability to handle both easy and hard problems.

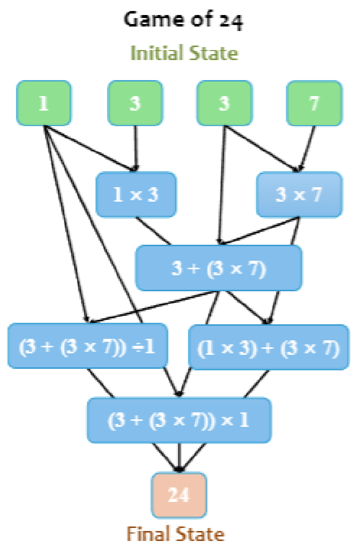


Figure 6.1.1: A thought structure for the Game of 24 [25].

6.1.2 Models

In this study, we prioritize the use of *smaller-scale LLMs*, ranging from 7 billion to 70 billion parameters. While recent advances in LLM development have been centered around massive models with hundreds of billions of parameters, there is increasing interest in exploring how well smaller models can perform on complex reasoning tasks. Smaller models, such as Llama2-7b and Mistral-7b, are faster and more computationally efficient, making them more practical for a wider range of applications. Moreover, testing advanced prompting techniques, such as Chain-of-Thought and Faithful-CoT, with these smaller models can reveal whether such methods can help bridge the gap between smaller and larger models, potentially unlocking emergent reasoning abilities without the need for massive parameter counts.

We rely on pre-trained models from Hugging Face, specifically choosing instruction-tuned variants designed to follow natural language instructions more effectively.

Llama2-7b-instruct

- **Llama2-7b-instruct** is a version of Meta's Llama2, fine-tuned for instruction-following tasks. The base Llama2 model was trained on a vast corpus of text data to improve language generation and understanding, with the 7-billion-parameter version being one of the smaller but highly effective variants. Fine-tuning it on instructional data allows it to better handle tasks like arithmetic problem-solving, reasoning, and answering structured prompts. For this study, we applied Llama2-7b-instruct specifically to the GSM8K and SVAMP datasets, given its demonstrated ability to perform well on language understanding and basic mathematical tasks.

- **Why smaller models?** While models with 7 billion parameters are significantly smaller than state-of-the-art models like GPT-4, they retain a strong balance between computational efficiency and performance, making them suitable for environments where resources are constrained. Smaller models like Llama2-7b also allow for rapid experimentation with various prompting techniques without incurring excessive computational costs, making them an ideal candidate for our study.

Llama3-8b-instruct

- **Llama3-8b-instruct** is the next evolution in the Llama series, with an increase in parameter count and enhanced capabilities in reasoning and instruction-following tasks. As an 8-billion-parameter model, it strikes a balance between size and efficiency, potentially improving over its predecessors in complex reasoning tasks like the Game of 24. Llama3 was trained on a diverse dataset, including larger volumes of technical and reasoning-heavy content, making it particularly suitable for multi-step reasoning problems.
- **Use case:** In this study, Llama3-8b-instruct is employed for RiddleSense and the Game of 24 dataset, where its enhanced reasoning abilities are tested. This is significant as it allows us to compare Llama3's performance against larger models like Llama3.1-70b, while applying techniques such as CoT and Self-Consistency.

Llama3.1-8b-instruct

- **Llama3.1-8b-instruct** builds upon Llama3, featuring improvements in model architecture, data preprocessing, and fine-tuning techniques. With the same parameter count as Llama3-8b, Llama3.1-8b aims to further refine the model's ability to follow complex instructions and handle nuanced reasoning tasks. Instruction-tuning for this version has been particularly focused on mathematical reasoning and code understanding, making it a strong candidate for solving arithmetic puzzles and logic problems.
- **Use case:** Similar to Llama3-8b-instruct, Llama3.1-8b-instruct is used for RiddleSense and Game of 24, enabling us to assess the improvements brought by the newer version. This model's performance will be directly compared with that of Llama3-8b and other instruction-tuned models, to evaluate how incremental changes in architecture and training data impact performance in complex problem-solving tasks.

Llama3.1-70b-instruct

- **Llama3.1-70b-instruct** represents a much larger model in the Llama family, featuring 70 billion parameters. The dramatic increase in parameters allows this model to handle even more intricate reasoning, problem-solving, and natural language understanding tasks. Instruction-tuning further improves its performance on tasks that require multiple reasoning steps, such as arithmetic puzzles or games requiring logical deductions.
- **Use case:** This model is also deployed on RiddleSense and on Game of 24 dataset to test whether scaling up the number of parameters can significantly enhance performance. Given that smaller models are often constrained by their ability to generalize across complex tasks, Llama3.1-70b-instruct will be used as a benchmark to assess the impact of model size on puzzle-solving accuracy, especially when paired with advanced prompting methods like CoT, Self-Consistency, and puzzle translation into Python code.

Mistral-7b-instruct-v0.3

- **Mistral-7b-instruct-v0.3** is a smaller model within the Mistral family, designed for efficient reasoning and instruction-following tasks. Despite its relatively modest size of 7 billion parameters, Mistral-7b has been shown to achieve competitive performance with larger models due to its optimized architecture and high-quality pre-training data. The instruction-tuned variant has been fine-tuned for tasks like logical reasoning, problem-solving, and arithmetic, making it an ideal candidate for evaluating puzzles like the Game of 24.

- **Use case: Mistral-7b-instruct-v0.3** is used alongside Llama3.1-70b and Llama3-8b for the two puzzle datasets. By including this smaller but efficient model, we aim to investigate whether advanced prompting methods can compensate for the reduced parameter count, and how Mistral compares to similarly sized models like Llama2-7b.

Model Architecture Comparison: Llama vs Mistral

While both Mistral and Llama models are designed for natural language understanding and reasoning tasks, there are key architectural differences that could influence their performance on complex tasks like puzzle-solving.

Llama models (Llama2, Llama3 and Llama3.1 series) follow a more traditional transformer architecture with optimizations aimed at scaling up the number of parameters while maintaining training efficiency. Llama models are known for their relatively high-quality pre-training, which enables strong performance even at smaller parameter scales like 7b and 8b. Llama models prioritize language generation and reasoning, but their performance scales significantly with size, meaning larger models such as Llama3.1-70b can handle more complex, multi-step reasoning tasks compared to the smaller variants.

Mistral models, on the other hand, introduce a more efficient transformer architecture that focuses on improved **weight-sharing mechanisms** and enhanced attention mechanisms to increase performance without needing as many parameters. Despite its 7b parameter count, Mistral-7b has been designed with architectural improvements that help it match or exceed the performance of larger models in certain tasks, particularly when combined with optimized prompting techniques. The goal with Mistral's architecture is to achieve similar or better reasoning capabilities while maintaining a more lightweight and resource-efficient design.

By comparing these two approaches, our experiments aim to determine whether architectural innovations like those in Mistral enable it to close the performance gap with larger Llama models when tackling complex reasoning tasks like puzzle solving and math word problems.

Previous Use of Models

While the math datasets (GSM8K and SVAMP) and RiddleSense have been used extensively in evaluating various models, the Game of 24 has primarily been evaluated with large models such as GPT-3.5, GPT-4. By incorporating smaller models like Llama3.1-70b and Mistral-7b, our experiments explore how well these models perform on reasoning tasks when advanced prompting methods are applied. This also opens up the possibility of discovering whether smaller models can handle complex tasks as effectively as larger ones through innovative approaches like puzzle translation.

6.1.3 Methodologies

This section outlines the different methodologies employed in our experiments across the selected datasets. We utilized various prompting techniques and reasoning methods to evaluate the performance of small LLMs in math reasoning tasks (GSM8K, SVAMP) and puzzle-solving (RiddleSense, Game of 24). The methods we applied include Standard Input-Output (IO) prompting, Chain-of-Thought (CoT) prompting, Least-to-Most (LtM) prompting, and puzzle translation, with Faithful-CoT for the math datasets and a specialized adaptation of Faithful-CoT used for translating natural language into Python code for Game of 24.

Standard IO Prompting

Standard IO prompting involves providing the model with a question and its corresponding answer in the context. The model then attempts to replicate this pattern for new inputs. This method serves as a simple baseline and is commonly used in a variety of reasoning tasks. For both math and puzzle datasets, IO prompting presents the problem in natural language and expects the model to output a solution without detailing intermediate reasoning steps.

In the context of Game of 24, the model is provided with the four numbers and directly produces an equation that equals 24. For example, given the input "4, 9, 10, 13" the model might output something like:

$$(13 - 9) * (10 - 4) = 24$$

Chain-of-Thought Prompting

CoT prompting enhances LLM reasoning by breaking down the problem into intermediate steps. Rather than outputting the final answer immediately, the model is prompted to articulate the thought process involved in solving the problem. This approach is especially useful for tasks that require multi-step reasoning, making it effective for both math problems and puzzles that involve complex operations.

For instance, in GSM8K, CoT might involve showing the intermediate calculations needed to arrive at the solution, while in Game of 24, it could display the step-by-step application of arithmetic operations to the four input numbers.

Least-to-Most Prompting

LtM prompting [143], is designed to handle complex problems by decomposing them into a series of smaller, simpler subproblems. Unlike CoT, LtM explicitly simplifies the initial problem into individual steps before solving each part in a progressive manner. This method has been particularly useful in solving math problems, where complex questions can be broken down into basic arithmetic operations or logical subcomponents.

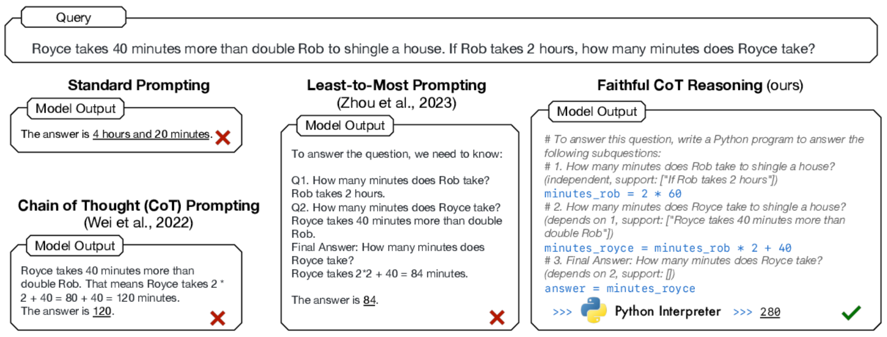
For example, in a word problem from SVAMP, LtM would first isolate simple operations—like subtraction or multiplication—and solve them step by step, ultimately arriving at the final answer. This method, however, is not applied to puzzle datasets in our experiments.

Faithful-CoT (for Math Datasets)

Faithful-CoT is a two-stage pipeline introduced to combine both natural language (NL) reasoning and symbolic language (SL) execution. In this approach, the model first generates a reasoning chain interleaving natural language descriptions and symbolic computations (such as Python code). The second stage involves using an external solver (e.g. a Python interpreter) to execute the symbolic code and compute the final answer.

In the GSM8K and SVAMP datasets, Faithful-CoT plays a crucial role in ensuring that the final answer is derived accurately through a deterministic process. For example, a complex math word problem might be translated into a series of Python operations that are then executed by an external solver to produce the answer. This guarantees correctness and transparency in the solution process.

Faithful-CoT's design makes it a powerful tool for tasks that require precise arithmetic or logical operations, providing reliable solutions through the execution of generated code.



Royce takes 40 minutes more than double Rob to shingle a house. If Rob takes 2 hours, how many minutes does Royce take?

Standard Prompting

The answer is 4 hours and 20 minutes. ✗

Chain of Thought (CoT) Prompting (Wei et al., 2022)

Royce takes 40 minutes more than double Rob. That means Royce takes 2 * 2 + 40 = 80 + 40 = 120 minutes.
The answer is 120. ✗

Least-to-Most Prompting (Zhou et al., 2023)

To answer the question, we need to know:

Q1. How many minutes does Rob take?
Rob takes 2 hours.
Q2. How many minutes does Royce take?
Royce takes 40 minutes more than double Rob.
Final Answer: How many minutes does Royce take?
Royce takes 2*2 + 40 = 84 minutes.

The answer is 84. ✗

Faithful CoT Reasoning (ours)

```
# To answer this question, write a Python program to answer the following subquestions:
# 1. How many minutes does Rob take to shingle a house? (independent, support: ["If Rob takes 2 hours"])
minutes_rob = 2 * 60
# 2. How many minutes does Royce take to shingle a house? (depends on 1, support: ["Royce takes 40 minutes more than double Rob"])
minutes_royce = minutes_rob * 2 + 40
# 3. Final Answer: How many minutes does Royce take? (depends on 2, support: [])
answer = minutes_royce
>>> Python Interpreter >>> 280
```
✓

Figure 6.1.2: A sample output for a math question from the four above methods. The ground-truth answer is 280. [78]

Puzzle Translation (for Puzzle Datasets)

For RiddleSense, puzzle translation from natural language into code is a method that doesn't make much sense. For instance, for the riddle:

I have five fingers, but I am not alive. What am I?
(A) piano (B) computer (C) ***glove*** *(D) claw (E) hand*

there is no clear way on how it can be described in Python code. For that reason, we do not apply puzzle translation for RiddleSense.

For the Game of 24, we employed a method that draws inspiration from both Faithful-CoT and Program of Thoughts, where the model translates the natural language input into Python code that is then executed to solve the puzzle. This neurosymbolic approach leverages the LLM's ability to interpret the task and translate it into a form suitable for external computation.

6.1.4 Prompting Settings

Zero-shot and Few-shot Setup

For all methods applied for the math reasoning datasets, a 8-shot setting has been applied, meaning that 8 (random) examples have been given to the model in the user prompt. For these datasets we have repeated the experiments with and without the use of a system prompt (an initial prompt that gives the models the instructions describing the way we want it to answer).

For the puzzle datasets we have the following settings:

- RiddleSense: For both IO Standard and CoT prompting we have tested zero shot, 5-shot and 8-shot examples.
- Game of 24: For IO Standard and CoT prompting we have tested zero-shot, 5-shot and 8-shot (random) exmpales, while for puzzle translation we have only tested the zero shot setting. For the latest method, we have also tried to indicate the reasoning steps that should be followed.

The prompts for each case are given at the Appendix 8.1.

Self-Consistency

For math datasets, Self-Consistency has also been tested for all methods, with the use of a system prompt. The number of generated chains for the self-consistency method is n=5, and the most consistent/voted outcome is the final given answer.

For the RiddleSense we have tested Self-Consistency for both IO Standard prompting and CoT and in each case we have used the few-shot setting that had the best results in the most cases. For example for the IO Standard prompting we have tested Self-Consistency on the 8-shot setting, while for the CoT prompting we have tested Self-Consistency on the 5-shot setting. A number of branches/chains equal to n=5 is used here. Simillarly, we have applied Self-Consistency for the Game of 24, testing also a number of braches equal to n=10.

Prompting Parameters

- **Temperature:** For every method besides neurosymbolic techniques and Self-Consistency settings, we have used a temperature equal to 0. In the case of Self-Consistency, for math datasets we have used a temperature equal to 0.4 for every method, and for puzzle datasets we have used a temperature equal to 0.7 for every method. Finally, in the case of Game of 24 with puzzle translation we have also used a temperature value equal to 0.7.
- **Maximum Tokens:** For the maximum tokens parameter we have used in every case, a value equal to 500, except for the puzzle translation methods, where this value is equal to 1000.

Note: The above values are applied to all tested models in each case.

6.2 Experimental Results

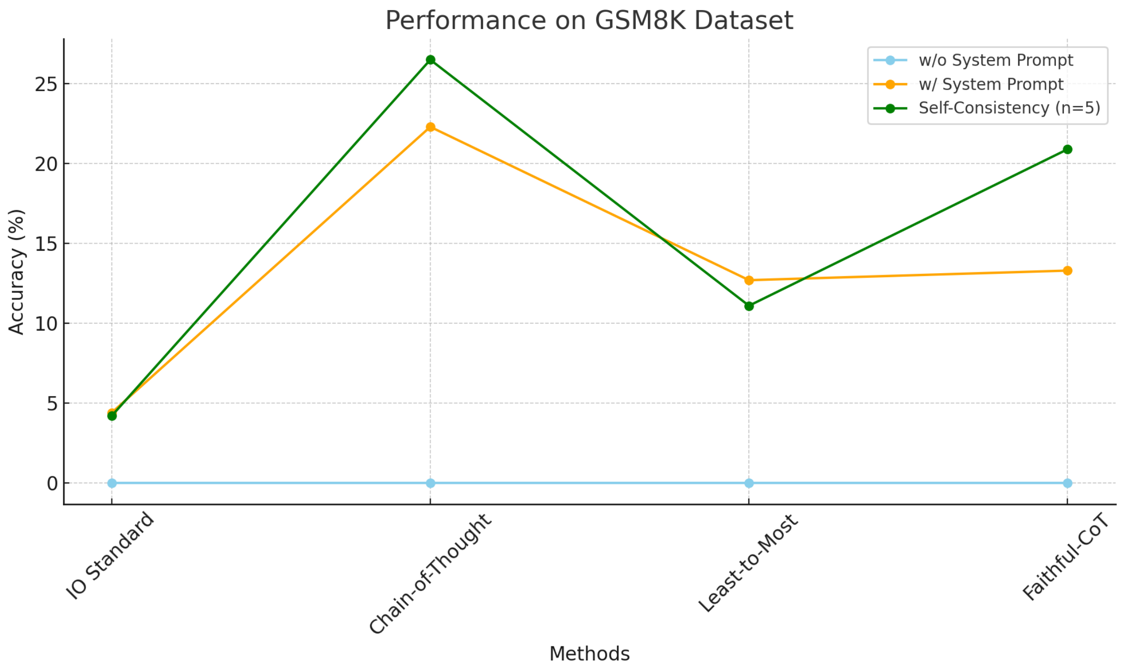
This section presents the performance of various prompting methods and model configurations across four datasets: GSM8K, SVAMP, RiddleSense, and Game of 24. Each dataset illustrates unique aspects of reasoning and problem-solving capabilities in language models, comparing standard input-output (IO), Chain-of-Thought (CoT), Least-to-Most (LtM), Faithful-CoT, and Puzzle Translation approaches across different prompting configurations (zero-shot, few-shot, and self-consistency settings).

6.2.1 GSM8K Results

The performance results for the GSM8K dataset are shown below:

Methods	**w/o System Prompt**	**w/ System Prompt**	**Self-Consistency (n=5)**
IO Standard	0%	4.4%	4.2%
Chain-of-Thought	0%	22.3%	26.5%
Least-to-Most	0%	12.7%	11.1%
Faithful-CoT	0%	13.3%	20.9%

Table 6.1: GSM8K Results



Observations:

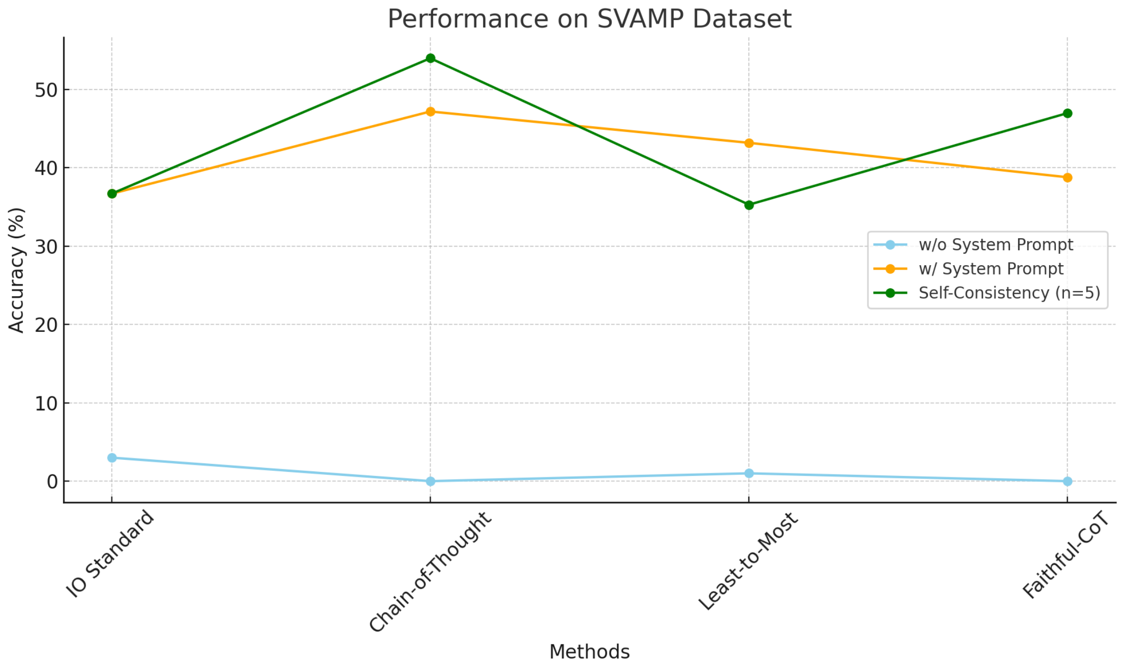
- The lack of a system prompt leads to 0% accuracy for all applied methods. Based on our observations, this has happened due to the "overfitting" of the model to the 8 given examples. Llama2-7b-instruct was repeating the examples given instead of answering the final question.
- The addition of a system prompt has improved the results, but the model still underperforms in the task, with a maximum accuracy in the CoT-SC setting.
- CoT was proved the most succesful method, while Self-Consistency improved the results for CoT and Faithful-CoT.
- In contrast to the original paper of Faithful-CoT [78], where models like GPT-3.5 and GPT-4 were utilized, CoT achieved better results than the neurosymbolic method with the Llama2-7b model.

6.2.2 SVAMP Results

The performance results for the SVAMP dataset are shown below:

Methods	**w/o System Prompt**	**w/ System Prompt**	**Self-Consistency (n=5)**
IO Standard	3%	36.7%	36.7%
Chain-of-Thought	0%	47.2%	54.0%
Least-to-Most	1%	43.2%	35.3%
Faithful-CoT	0%	38.8%	47.0%

Table 6.2: SVAMP Results



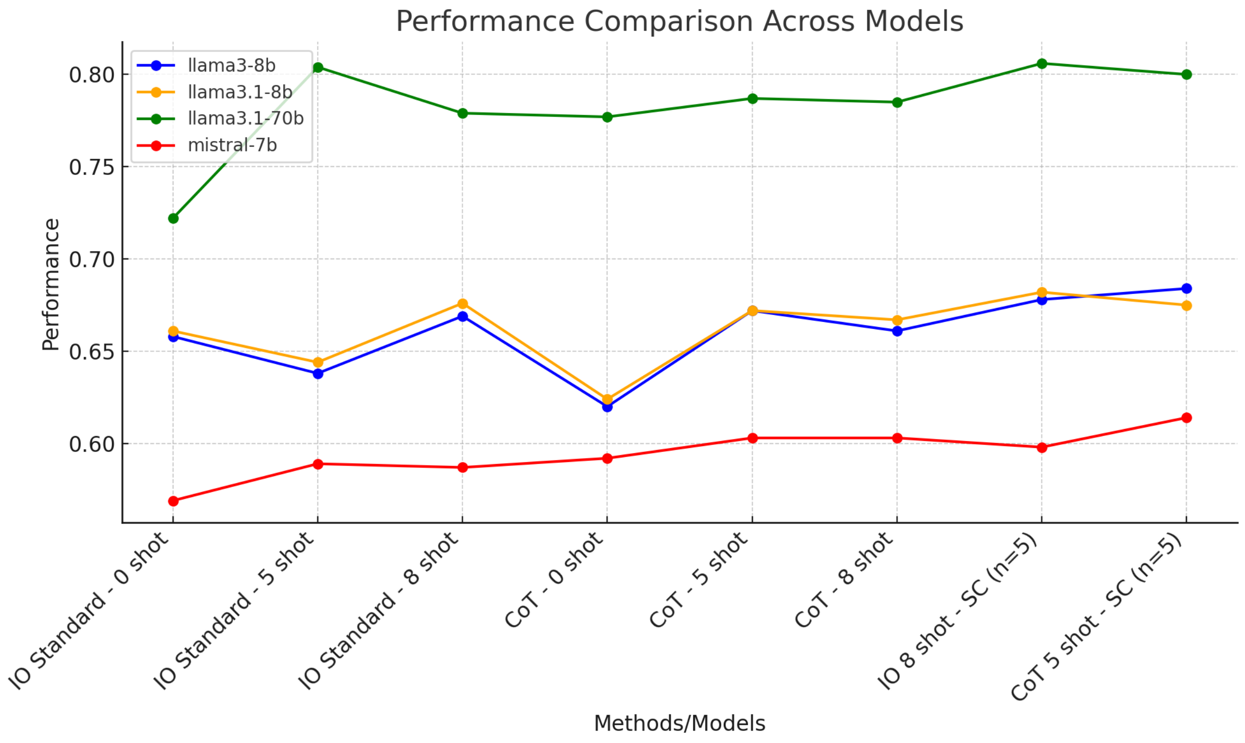
Observations: The observations for the results on the SVAMP dataset are very similar to the ones on the GSM8K. The only difference is that the accuracy of most of the methods in this dataset is better than the results seen previously. This is a general trend on the available leader boards and is likely because the problems presented in SVAMP are slightly easier compared to GSM8K.

6.2.3 RiddleSense Results

The performance results for the RiddleSense dataset are shown below:

Methods/Models	llama3-8b	llama3.1-8b	llama3.1-70b	mistral-7b
IO Standard - 0 shot	0.658	0.661	0.722	0.569
IO Standard - 5 shot	0.638	0.644	0.804	0.589
IO Standard - 8 shot	0.669	0.676	0.779	0.587
CoT - 0 shot	0.620	0.624	0.777	0.592
CoT - 5 shot	0.672	0.672	0.787	0.603
CoT - 8 shot	0.661	0.667	0.785	0.603
IO 8 shot - SC (n=5)	0.678	0.682	0.806	0.598
CoT 5 shot - SC (n=5)	0.684	0.675	0.8	0.614

Table 6.3: RiddleSense Results



Observations:

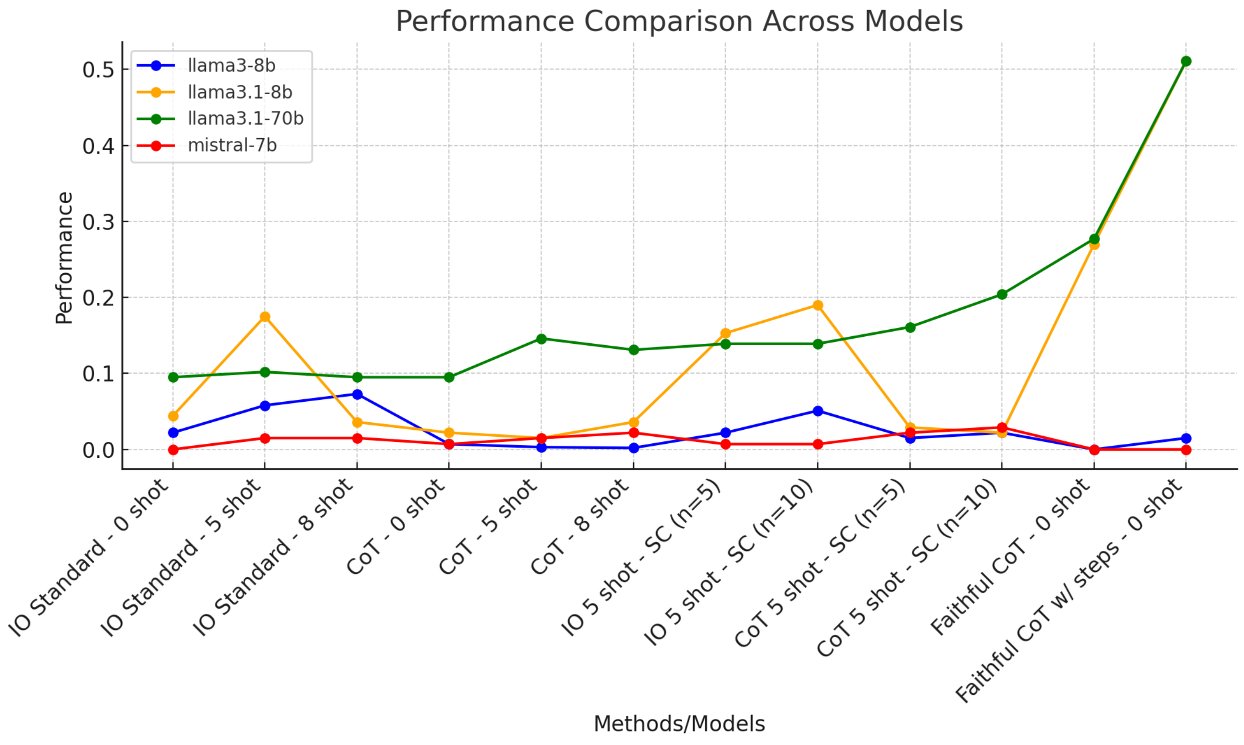
- Firstly, it is observed that the most important factor for achieving better results in RiddleSense is the size of the model (its number of parameters), with llama3.1-70b-instruct to consistently outperform the other models, reaching up to 0.806 in accuracy with IO Standard 8-shot prompting with Self-Consistency. On the other hand, mistral-7b-v0.3, while efficient, struggles to match the performance of llama models across all configurations underscoring the importance of model scale for riddles.
- Few-shot prompting improves the results for both IO Standard and CoT prompting, with the shot count not being clearly better between 5 and 8.
- Self-consistency (n=5) generally enhanced the results for all models.

6.2.4 Game-of-24 Results

The performance results for the Game of 24 task are shown below:

Methods/Models	**llama3-8b**	**llama3.1-8b**	**llama3.1-70b**	**mistral-7b**
IO Standard - 0 shot	0.022	0.044	0.095	0.0
IO Standard - 5 shot	0.058	0.175	0.102	0.015
IO Standard - 8 shot	0.073	0.036	0.095	0.015
CoT - 0 shot	0.007	0.022	0.095	0.007
CoT - 5 shot	0.003	0.015	0.146	0.015
CoT - 8 shot	0.002	0.036	0.131	0.022
IO 5 shot - SC (n=5)	0.022	0.153	0.139	0.007
IO 5 shot - SC (n=10)	0.051	0.19	0.139	0.007
CoT 5 shot - SC (n=5)	0.015	0.029	0.161	0.022
CoT 5 shot - SC (n=10)	0.022	0.022	0.204	0.029
Faithful CoT - 0 shot	0.0	0.27	0.277	0.0
Faithful CoT w/ steps - 0 shot	0.015	0.511	0.511	0.0

Table 6.4: Game-of-24 Results



Observations:

- Again the model's size plays a crucial role in the performance, with llama3.1-70b achieving the best results for most of the cases and mistral-7b struggling in this high-complexity environment, with most configurations failing to exceed minimal accuracy.
- Increasing the number of shots does not consistently imporve results for IO and CoT.
- In some cases, CoT not only doesn't enhance the performance but it even generates worse results than Standard IO prompting. Also, Self-Consistency has helped some models to perform better, but this was not always the case.
- Regarding the puzzle translation technique, it is observed that mistral-7b and llama3-8b are not able to generate executable code, thus leading to 0% results. On the other hand, llama3.1 models were

able to generate executable code and they achieved the best accuracy results compared to every other experiment. Also the addition of the steps of code that the model should generate, further enhanced the results for these models. Finally, even if llama3.1 achieved the best results with the neurosymbolic method, the increased number of parameters of the llama3.1-70b model didn't lead to a further increased accuracy.

Chapter 7

Conclusion

7.1 Discussion

This thesis has explored the capacity of large language models (LLMs) to engage in complex reasoning and problem-solving across a variety of puzzle and mathematical reasoning tasks. By categorizing puzzles into rule-based and rule-less types, this work has provided a structured approach to understanding the diverse cognitive demands that different puzzles place on LLMs. Through a comprehensive survey of methods, datasets, and benchmarks, the thesis also highlights the current landscape of LLM performance in puzzle solving, along with the challenges these models face.

Our experiments were designed to delve deeper into the practical application of various prompting strategies and techniques across four datasets—GSM8K, SVAMP, RiddleSense, and Game of 24. For the mathematical datasets (GSM8K and SVAMP), the methods explored included standard input-output (IO) prompting, Chain-of-Thought (CoT), Least-to-Most (LtM) prompting, and Faithful-CoT, evaluated under zero-shot, few-shot, and self-consistency settings. For the puzzle datasets (RiddleSense and Game of 24), we compared IO, CoT, and a translation-based approach akin to Faithful-CoT, which translates natural language into executable code.

The survey conducted in this thesis provides insights into the application of various prompting techniques and model fine-tuning across puzzle types. Techniques like few-shot prompting, Chain-of-Thought (CoT), and fine-tuning were broadly used in both rule-based and rule-less categories, with puzzle-solving methods like ToT and GoT enhancing complex reasoning in deterministic puzzles. However, these techniques require a high number of LLM invocations, which can limit scalability. The survey also highlights that neuro-symbolic techniques, like translating natural language into code, are largely unexplored in puzzle-solving contexts. Furthermore, the lack of variety in methods and datasets for rule-based stochastic and rule-less programming puzzles presents an opportunity for future research.

The experimental findings reinforced some of the survey's insights, while revealing unique challenges across datasets. In the math datasets (GSM8K and SVAMP), the absence of system prompts led to a performance drop, confirming the importance of structured prompts for mathematical reasoning. Faithful-CoT did not yield improvements on Llama2-7b-instruct, and although CoT with Self-Consistency was the most effective approach, even basic problems posed challenges, suggesting limitations in current reasoning capabilities of smaller LLMs.

For RiddleSense, model size emerged as the key factor in performance, with the largest model, Llama3.1-70b, performing best, while CoT alone did not guarantee success, hinting at the need for more sophisticated approaches beyond reasoning chains alone. Similarly, in the Game of 24 dataset, only Llama3.1 models were able to generate executable code with the Faithful-CoT method, emphasizing the compatibility between certain methods and model architecture. Additionally, CoT did not consistently enhance results, suggesting that reasoning chains may have limited applicability for puzzles requiring precise computational steps.

These experimental observations illustrate that while LLMs can tackle complex reasoning to an extent, challenges persist, especially in mathematical and rule-based problem-solving where smaller models often lack the robustness to execute reliable reasoning consistently.

7.2 Future Work

For future work, several key areas offer promising directions for advancing research in LLM-based puzzle solving:

- **Methodological Improvements:** A primary area for future research lies in leveraging neuro-symbolic techniques, particularly natural language to code translation. This approach, largely unexplored in puzzle contexts, could bridge the gap between abstract reasoning and executable solutions, especially for complex puzzles requiring precise operations. Additionally, there is potential in evaluating advanced prompting topologies, such as graph- or tree-based reasoning structures, which could enhance the reasoning depth of smaller LLMs while maintaining computational efficiency.
- **Dataset Creation:** The survey highlighted gaps in available datasets, especially for stochastic puzzles and programming challenges. Developing new benchmarks for these underrepresented categories could stimulate research in these areas, offering a wider array of tasks to test and expand the reasoning capabilities of LLMs.
- **Puzzle Generation:** Research into automatic puzzle generation has been limited, with recent initiatives like RISCORE [92] just beginning to address this field. Since understanding and solving puzzles is foundational to generating them, advancements in LLM reasoning abilities could directly impact puzzle generation research. This area holds particular promise for developing models capable of creating novel puzzles that test diverse aspects of reasoning, logic, and inference, thus adding a new dimension to the evaluation of LLMs.

Chapter 8

Appendices

Contents

8.1 Prompts used for the experiments

8.1.1 GSM8K

Standard IO Prompting

System Prompt

"You are a chatbot that always responds in the way of 'The answer is X', where X is just a number, the answer to the question."

User Prompt

"Q: There are 15 trees in the grove. Grove workers will plant trees in the grove today. After they are done, there will be 21 trees. How many trees did the grove workers plant today?

A: The answer is 6.

Q: If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot?

A: The answer is 5.

Q: Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total?

A: The answer is 39.

Q: Jason had 20 lollipops. He gave Denny some lollipops. Now Jason has 12 lollipops. How many lollipops did Jason give to Denny?

A: The answer is 8.

Q: Shawn has five toys. For Christmas, he got two toys each from his mom and dad. How many toys does he have now?

A: The answer is 9.

Q: There were nine computers in the server room. Five more computers were installed each day, from monday to thursday. How many computers are now in the server room?

A: The answer is 29.

Q: Michael had 58 golf balls. On tuesday, he lost 23 golf balls. On wednesday, he lost 2 more. How many golf balls did he have at the end of wednesday?

A: The answer is 33.

Q: Olivia has $23. She bought five bagels for $3 each. How much money does she have left?

A: The answer is 8.

Q: [QUESTION]

A:"

Chain-of-Thought (CoT) Prompting

System Prompt

"You are an assistant proficient in solving math problems using a step-by-step reasoning method. Follow the format shown in the examples provided and solve only the last question step-by-step. Give the final answer in the way of 'The answer is X', where X is just a number."

User Prompt

"Q: There are 15 trees in the grove. Grove workers will plant trees in the grove today. After they are done, there will be 21 trees. How many trees did the grove workers plant today?

A: We start with 15 trees. Later we have 21 trees. The difference must be the number of trees they planted. So, they must have planted 21 - 15 = 6 trees.

The answer is 6.

Q: If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot?

A: There are 3 cars in the parking lot already. 2 more arrive. Now there are 3 + 2 = 5 cars.

The answer is 5.

Q: Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total?

A: Leah had 32 chocolates and Leah's sister had 42. That means there were originally 32 + 42 = 74 chocolates. 35 have been eaten. So in total they still have 74 - 35 = 39 chocolates.

The answer is 39.

Q: Jason had 20 lollipops. He gave Denny some lollipops. Now Jason has 12 lollipops. How many lollipops did Jason give to Denny?

A: Jason had 20 lollipops. Since he only has 12 now, he must have given the rest to Denny. The number of lollipops he has given to Denny must have been 20 - 12 = 8 lollipops.

The answer is 8.

Q: Shawn has five toys. For Christmas, he got two toys each from his mom and dad. How many toys does he have now?

A: He has 5 toys. He got 2 from mom, so after that he has 5 + 2 = 7 toys. Then he got 2 more from dad, so in total he has 7 + 2 = 9 toys.

The answer is 9.

Q: There were nine computers in the server room. Five more computers were installed each day, from monday to thursday. How many computers are now in the server room?

A: There are 4 days from monday to thursday. 5 computers were added each day. That means in total 4 * 5 = 20 computers were added. There were 9 computers in the beginning, so now there are 9 + 20 = 29 computers.

The answer is 29.

Q: Michael had 58 golf balls. On tuesday, he lost 23 golf balls. On wednesday, he lost 2 more. How many golf balls did he have at the end of wednesday?

A: Michael initially had 58 balls. He lost 23 on Tuesday, so after that he has 58 - 23 = 35 balls. On Wednesday he lost 2 more so now he has 35 - 2 = 33 balls.

The answer is 33.

Q: Olivia has $23. She bought five bagels for $3 each. How much money does she have left?

A: She bought 5 bagels for $3 each. This means she spent 5 * $3 = $15 on the bagels. She had $23 in beginning, so now she has $23 - $15 = $8.

The answer is 8.

Q: [QUESTION] A:"

Least-to-Most Prompting

System Prompt

"You are an assistant proficient in solving math problems using a Least-to-Most (LTM) reasoning method. For each question, you will break down the problem into smaller subquestions Q1, Q2, etc., and then provide the final answer in the format: 'The answer is X', where X is just a number. Follow the format shown in the examples provided and answer only the final question."

User Prompt

Q: There are 15 trees in the grove. Grove workers will plant trees in the grove today. After they are done, there will be 21 trees. How many trees did the grove workers plant today?

A: To answer the above question, we need to answer the following subquestions:

Q1. How many trees are there in the beginning? There are 15 trees.

Q2. How many trees will there be after the grove workers plant trees? There are 21 trees.

Final Answer: How many trees did the grove workers plant today? They planted 21 - 15 = 6 trees. The answer is 6.

Q: If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot?

A: To answer the above question, we need to answer the following subquestions:

Q1. How many cars are there in the beginning? There are 3 cars.

Q2. How many cars arrive? 2 more cars arrive.

Final Answer: How many cars are in the parking lot? Now there are 3 + 2 = 5 cars. The answer is 5.

Q: Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total?

A: To answer the above question, we need to answer the following subquestions:

Q1. How many chocolates did Leah have in the beginning? Leah had 32 chocolates in the beginning.

Q2. How many chocolates did her sister have in the beginning? Her sister had 42 in the beginning.

Q3. How many chocolates did they eat? They ate 35 chocolates.

Final Answer: How many chocolates do they have left in total? They still have 32 + 42 - 35 = 39 chocolates left in total. The answer is 39.

Q: Jason had 20 lollipops. He gave Denny some lollipops. Now Jason has 12 lollipops. How many lollipops did Jason give to Denny?

A: To answer the above question, we need to answer the following subquestions:

Q1. How many lollipops did Jason have in the beginning? Jason had 20 lollipops in the beginning.

Q2. How many lollipops does Jason have now? Jason has 12 lollipops now.

Final Answer: How many lollipops did Jason give to Denny? He gave 20 - 12 = 8 lollipops to Denny. The answer is 8.

Q: Shawn has five toys. For Christmas, he got two toys each from his mom and dad. How many toys does he have now?

A: To answer the above question, we need to answer the following subquestions:

Q1. How many toys does Shawn have in the beginning? Shawn has 5 toys in the beginning.

Q2. How many toys did Shawn get from his mom? Shawn got 2 toys from his mom.

Q3. How many toys did Shawn get from his dad? Shawn got 2 toys from his dad.

Final Answer: How many toys does Shawn have now? Shawn has 5 + 2 + 2 = 9 toys now. The answer is 9.

Q: There were nine computers in the server room. Five more computers were installed each day, from monday to thursday. How many computers are now in the server room?

A: To answer the above question, we need to answer the following subquestions:

Q1. How many computers were there in the beginning? There were 9 computers in the beginning.

Q2. How many computers were added each day? 5 computers were added each day.

Q3. How many days were there from monday to thursday? There are 4 days from monday to thursday.

Final Answer: How many computers are now in the server room? Now there are 9 + 4*5 = 29 computers. The answer is 29.

Q: Michael had 58 golf balls. On tuesday, he lost 23 golf balls. On wednesday, he lost 2 more. How many golf balls did he have at the end of wednesday?

A: To answer the above question, we need to answer the following subquestions:

Q1. How many golf balls did Michael have in the beginning? Michael had 58 golf balls in the beginning.

Q2. How many golf balls did Michael lose on tuesday? Michael lost 23 golf balls on tuesday.

Q3. How many golf balls did Michael lose on wednesday? Michael lost 2 golf balls on wednesday.

Final Answer: How many golf balls did Michael have at the end of wednesday? He had 58 - 23 - 2 = 33 balls. The answer is 33.

Q: Olivia has $23. She bought five bagels for $3 each. How much money does she have left?

A: To answer the above question, we need to answer the following subquestions:

Q1. How much money did Olivia have in the beginning? Olivia had $23 in the beginning.

Q2. How much did each bagel cost? Each bagel cost $3.

Q3. How many bagels did Olivia buy? Olivia bought 5 bagels.

Final Answer: How much money does Olivia have left? Now she has $23 - $5*3 = $8. The answer is 8.

Q: [QUESTION]

A: To answer the above question, we need to answer the following subquestions:"

Faithful-CoT

User Prompt

"# Q: There are 15 trees in the grove. Grove workers will plant trees in the grove today. After they are done, there will be 21 trees. How many trees did the grove workers plant today?

# To answer this question, write a Python program to answer the following subquestions:

# 1. How many trees are there in the beginning? (independent, support: ["There are 15 trees"])

$trees\_begin = 15$

# 2. How many trees are there in the end? (independent, support: ["there will be 21 trees"])

$trees\_end = 21$

# 3. How many trees did the grove workers plant today? (depends on 1 and 2, support: [])

$trees\_today = trees\_end - trees\_begin$

# 4. Final Answer: How many trees did the grove workers plant today? (depends on 3, support: [])

$answer = trees\_today$

# Q: There were nine computers in the server room. Five more computers were installed each day, from monday to thursday. How many computers are now in the server room?

# To answer this question, write a Python program to answer the following subquestions:

# 1. How many computers were there in the beginning? (independent, support: ["There were nine computers"])

$computers\_begin = 9$

# 2. How many computers were installed each day? (independent, support: ["Five more computers were installed each day"])

$computers\_each\_day = 5$

# 3. How many days were there from monday to thursday? (independent, support: ["External knowledge: days of the week are Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday, so there are 4 days from Monday to Thursday"])

$days\_monday\_to\_thursday = 4$

# 4. How many new computers were installed? (depends on 2 and 3, support: [])

```
computers_new = 0
for i in range(days_monday_to_thursday):
    computers_new += computers_each_day
```

# 5. How many computers are now in the server room? (depends on 1 and 4, support: [])

$computers\_now = computers\_begin + computers\_new$

# 6. Final Answer: How many computers are now in the server room? (depends on 5, support: [])

$answer = computers\_now$

# Q: Danny has 3 bottles of soda. He drinks 90% of one bottle and gives 70% of the other two bottles to his friends. How much soda does Danny have left, expressed as a percentage of a bottle?

# To answer this question, write a Python program to answer the following subquestions:

# 1. How much of one bottle did Danny drink? (independent, support: ["He drinks 90% of one bottle"])

$fraction\_bottle\_drink = 0.9$

# 2. How much of the other two bottles did Danny give to his friends? (independent, support: ["gives 70% of the other two bottles to his friends"])

$fraction\_bottle\_give = 0.7$

# 3. How much soda is left in the bottle Danny drank from? (depends on 1, support: [])

$fraction\_bottle\_drink\_left = 1 - fraction\_bottle\_drink$

# 4. How much soda is left in the two bottles Danny gave his friends? (depends on 2, support: [])

$fraction\_bottle\_give\_left = (1 - fraction\_bottle\_give) * 2$

# 5. How much soda does Danny have left in total? (depends on 3 and 4, support: [])

$fraction\_soda\_left = fraction\_bottle\_drink\_left + fraction\_bottle\_give\_left$

# 6. How much soda does Danny have left in total, expressed as a percentage of a bottle? (depends on 5, support: ["External knowledge: fraction multiplied by 100 is a percent"])

$percent\_soda\_left = fraction\_soda\_left * 100$

# 7. Final Answer: How much soda does Danny have left, expressed as a percentage of a bottle? (depends on 6, support: [])

$answer = percent\_soda\_left$

# Q: In a fruit salad, there are raspberries, green grapes, and red grapes. There are seven more than 3 times the number of red grapes as green grapes. There are 5 less raspberries than green grapes. If there are 102 pieces of fruit in the salad, how many red grapes are in the salad?

# To answer this question, write a Python program to answer the following subquestions:

# 1. How many green grapes are there? (independent, support: [])

$green\_grapes = Symbol("green\_grapes")$

# 2. How many red grapes are there? (depends on 1, support: ["There are seven more than 3 times the number of red grapes as green grapes"])

$red\_grapes = 3 * green\_grapes + 7$

# 3. How many raspberries are there? (depends on 1, support: ["There are 5 less raspberries than green grapes"])

$raspberries = green\_grapes - 5$

# 4. How many total pieces of fruit are there? (depends on 1, 2, and 3, support: "there are 102 pieces of fruit in the salad")

$total\_fruit\_eq = Eq(green\_grapes + red\_grapes + raspberries, 102)$

# 5. How many green grapes are in the salad based on this equation? (depends on 1 and 4, support: [])

$green\_grapes\_val = solve\_it(total\_fruit\_eq, green\_grapes)[green\_grapes]$

# 6. How many red grapes are in the salad given the number of green grapes? (depends on 1, 2 and 5, support: "how many red grapes are in the salad?"])

$red\_grapes\_val = red\_grapes.subs(green\_grapes, green\_grapes\_val)$

# 7. Final Answer: how many red grapes are in the salad? (depends on 6, support: [])

$answer = red\_grapes\_val$

# Follow the format of the examples above to answer the following question. Don't say anything else at the beginning or at the end of your answer. Just give the code with the comments as in the examples.

# Q: [QUESTION]

# To answer this question, write a Python program to answer the following subquestions:"

8.1.2 SVAMP

Standard IO Prompting

System Prompt

"You are a chatbot that always responds in the way of 'The answer is X', where X is just a number, the answer to the question."

User Prompt

"Q: There are 15 trees in the grove. Grove workers will plant trees in the grove today. After they are done, there will be 21 trees. How many trees did the grove workers plant today?

A: The answer is 6.

Q: If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot?

A: The answer is 5.

Q: Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total?

A: The answer is 39.

Q: Jason had 20 lollipops. He gave Denny some lollipops. Now Jason has 12 lollipops. How many lollipops did Jason give to Denny?

A: The answer is 8.

Q: Shawn has five toys. For Christmas, he got two toys each from his mom and dad. How many toys does he have now?

A: The answer is 9.

Q: There were nine computers in the server room. Five more computers were installed each day, from monday to thursday. How many computers are now in the server room?

A: The answer is 29.

Q: Michael had 58 golf balls. On tuesday, he lost 23 golf balls. On wednesday, he lost 2 more. How many golf balls did he have at the end of wednesday?

A: The answer is 33.

Q: Olivia has $23. She bought five bagels for $3 each. How much money does she have left?

A: The answer is 8.

Q: [QUESTION]

A:"

Chain-of-Thought (CoT) Prompting

System Prompt

"You are an assistant proficient in solving math problems using a step-by-step reasoning method. Follow the format shown in the examples provided and solve only the last question step-by-step. Give the final answer in the way of 'The answer is X', where X is just a number."

User Prompt

"Q: There are 15 trees in the grove. Grove workers will plant trees in the grove today. After they are done, there will be 21 trees. How many trees did the grove workers plant today?

A: We start with 15 trees. Later we have 21 trees. The difference must be the number of trees they planted. So, they must have planted 21 - 15 = 6 trees. The answer is 6.

Q: If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot?

A: There are 3 cars in the parking lot already. 2 more arrive. Now there are 3 + 2 = 5 cars. The answer is 5.

Q: Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total?

A: Leah had 32 chocolates and Leah's sister had 42. That means there were originally 32 + 42 = 74 chocolates. 35 have been eaten. So in total they still have 74 - 35 = 39 chocolates. The answer is 39.

Q: Jason had 20 lollipops. He gave Denny some lollipops. Now Jason has 12 lollipops. How many lollipops did Jason give to Denny?

A: Jason had 20 lollipops. Since he only has 12 now, he must have given the rest to Denny. The number of lollipops he has given to Denny must have been 20 - 12 = 8 lollipops. The answer is 8.

Q: Shawn has five toys. For Christmas, he got two toys each from his mom and dad. How many toys does he have now?

A: He has 5 toys. He got 2 from mom, so after that he has 5 + 2 = 7 toys. Then he got 2 more from dad, so in total he has 7 + 2 = 9 toys. The answer is 9.

Q: There were nine computers in the server room. Five more computers were installed each day, from monday to thursday. How many computers are now in the server room?

A: There are 4 days from monday to thursday. 5 computers were added each day. That means in total 4 * 5 = 20 computers were added. There were 9 computers in the beginning, so now there are 9 + 20 = 29 computers. The answer is 29.

Q: Michael had 58 golf balls. On tuesday, he lost 23 golf balls. On wednesday, he lost 2 more. How many golf balls did he have at the end of wednesday?

A: Michael initially had 58 balls. He lost 23 on Tuesday, so after that he has 58 - 23 = 35 balls. On Wednesday he lost 2 more so now he has 35 - 2 = 33 balls. The answer is 33.

Q: Olivia has $23. She bought five bagels for $3 each. How much money does she have left?

A: She bought 5 bagels for $3 each. This means she spent 5 * $3 = $15 on the bagels. She had $23 in beginning, so now she has $23 - $15 = $8. The answer is 8.

Q: [QUESTION]

A:"

Least-to-Most Prompting

System Prompt

"You are an assistant proficient in solving math problems using a Least-to-Most (LTM) reasoning method. For each question, you will break down the problem into smaller subquestions Q1, Q2, etc., and then provide the final answer in the format: 'The answer is X', where X is just a number. Follow the format shown in the examples provided and answer only the final question."

User Prompt

"Q: There are 15 trees in the grove. Grove workers will plant trees in the grove today. After they are done, there will be 21 trees. How many trees did the grove workers plant today?

A: To answer the above question, we need to answer the following subquestions:

Q1. How many trees are there in the beginning? There are 15 trees.

Q2. How many trees will there be after the grove workers plant trees? There are 21 trees.

Final Answer: How many trees did the grove workers plant today? They planted 21 - 15 = 6 trees. The answer is 6.

Q: If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot?

A: To answer the above question, we need to answer the following subquestions:

Q1. How many cars are there in the beginning? There are 3 cars.

Q2. How many cars arrive? 2 more cars arrive.

Final Answer: How many cars are in the parking lot? Now there are 3 + 2 = 5 cars. The answer is 5.

Q: Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total?

A: To answer the above question, we need to answer the following subquestions:

Q1. How many chocolates did Leah have in the beginning? Leah had 32 chocolates in the beginning.

Q2. How many chocolates did her sister have in the beginning? Her sister had 42 in the beginning.

Q3. How many chocolates did they eat? They ate 35 chocolates.

Final Answer: How many chocolates do they have left in total? They still have 32 + 42 - 35 = 39 chocolates left in total. The answer is 39.

Q: Jason had 20 lollipops. He gave Denny some lollipops. Now Jason has 12 lollipops. How many lollipops did Jason give to Denny?

A: To answer the above question, we need to answer the following subquestions:

Q1. How many lollipops did Jason have in the beginning? Jason had 20 lollipops in the beginning.

Q2. How many lollipops does Jason have now? Jason has 12 lollipops now.

Final Answer: How many lollipops did Jason give to Denny? He gave 20 - 12 = 8 lollipops to Denny. The answer is 8.

Q: Shawn has five toys. For Christmas, he got two toys each from his mom and dad. How many toys does he have now?

A: To answer the above question, we need to answer the following subquestions:

Q1. How many toys does Shawn have in the beginning? Shawn has 5 toys in the beginning.

Q2. How many toys did Shawn get from his mom? Shawn got 2 toys from his mom.

Q3. How many toys did Shawn get from his dad? Shawn got 2 toys from his dad. Final Answer: How many toys does Shawn have now? Shawn has 5 + 2 + 2 = 9 toys now. The answer is 9.

Q: There were nine computers in the server room. Five more computers were installed each day, from monday to thursday. How many computers are now in the server room?

A: To answer the above question, we need to answer the following subquestions:

Q1. How many computers were there in the beginning? There were 9 computers in the beginning.

Q2. How many computers were added each day? 5 computers were added each day.

Q3. How many days were there from monday to thursday? There are 4 days from monday to thursday.

Final Answer: How many computers are now in the server room? Now there are 9 + 4*5 = 29 computers. The answer is 29.

Q: Michael had 58 golf balls. On tuesday, he lost 23 golf balls. On wednesday, he lost 2 more. How many golf balls did he have at the end of wednesday?

A: To answer the above question, we need to answer the following subquestions:

Q1. How many golf balls did Michael have in the beginning? Michael had 58 golf balls in the beginning.

Q2. How many golf balls did Michael lose on tuesday? Michael lost 23 golf balls on tuesday.

Q3. How many golf balls did Michael lose on wednesday? Michael lost 2 golf balls on wednesday.

Final Answer: How many golf balls did Michael have at the end of wednesday? He had 58 - 23 - 2 = 33 balls. The answer is 33.

Q: Olivia has $23. She bought five bagels for $3 each. How much money does she have left?

A: To answer the above question, we need to answer the following subquestions:

Q1. How much money did Olivia have in the beginning? Olivia had $23 in the beginning.

Q2. How much did each bagel cost? Each bagel cost $3.

Q3. How many bagels did Olivia buy? Olivia bought 5 bagels.

Final Answer: How much money does Olivia have left? Now she has $23 - $5*3 = $8. The answer is 8.

Q: [QUESTION]

A: To answer the above question, we need to answer the following subquestions:"

Faithful-CoT

User Prompt

"# Q: There are 15 trees in the grove. Grove workers will plant trees in the grove today. After they are done, there will be 21 trees. How many trees did the grove workers plant today?

# To answer this question, write a Python program to answer the following subquestions:

# 1. How many trees are there in the beginning? (independent, support: ["There are 15 trees"])

$trees\_begin = 15$

# 2. How many trees are there in the end? (independent, support: ["there will be 21 trees"])

$trees\_end = 21$

# 3. How many trees did the grove workers plant today? (depends on 1 and 2, support: [])

$trees\_today = trees\_end - trees\_begin$

# 4. Final Answer: How many trees did the grove workers plant today? (depends on 3, support: [])

$answer = trees\_today$

# Q: There were nine computers in the server room. Five more computers were installed each day, from monday to thursday. How many computers are now in the server room?

# To answer this question, write a Python program to answer the following subquestions:

# 1. How many computers were there in the beginning? (independent, support: ["There were nine computers"])

$computers\_begin = 9$

# 2. How many computers were installed each day? (independent, support: ["Five more computers were installed each day"])

$computers\_each\_day = 5$

# 3. How many days were there from monday to thursday? (independent, support: ["External knowledge: days of the week are Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday, so there are 4 days from Monday to Thursday"])

$days\_monday\_to\_thursday = 4$

# 4. How many new computers were installed? (depends on 2 and 3, support: [])

```
computers_new = 0
for i in range(days_monday_to_thursday):
    computers_new += computers_each_day
```

# 5. How many computers are now in the server room? (depends on 1 and 4, support: [])

$computers\_now = computers\_begin + computers\_new$

# 6. Final Answer: How many computers are now in the server room? (depends on 5, support: [])

$answer = computers\_now$

# Q: Danny has 3 bottles of soda. He drinks 90% of one bottle and gives 70% of the other two bottles to his friends. How much soda does Danny have left, expressed as a percentage of a bottle?

# To answer this question, write a Python program to answer the following subquestions:

# 1. How much of one bottle did Danny drink? (independent, support: ["He drinks 90% of one bottle"])

$fraction\_bottle\_drink = 0.9$

# 2. How much of the other two bottles did Danny give to his friends? (independent, support: ["gives 70% of the other two bottles to his friends"])

$fraction\_bottle\_give = 0.7$

# 3. How much soda is left in the bottle Danny drank from? (depends on 1, support: [])

$fraction\_bottle\_drink\_left = 1 - fraction\_bottle\_drink$

# 4. How much soda is left in the two bottles Danny gave his friends? (depends on 2, support: [])

$fraction\_bottle\_give\_left = (1 - fraction\_bottle\_give) * 2$

# 5. How much soda does Danny have left in total? (depends on 3 and 4, support: [])

$fraction\_soda\_left = fraction\_bottle\_drink\_left + fraction\_bottle\_give\_left$

# 6. How much soda does Danny have left in total, expressed as a percentage of a bottle? (depends on 5, support: ["External knowledge: fraction multiplied by 100 is a percent"])

$percent\_soda\_left = fraction\_soda\_left * 100$

# 7. Final Answer: How much soda does Danny have left, expressed as a percentage of a bottle? (depends on 6, support: [])

$answer = percent\_soda\_left$

# Q: In a fruit salad, there are raspberries, green grapes, and red grapes. There are seven more than 3 times the number of red grapes as green grapes. There are 5 less raspberries than green grapes. If there are 102 pieces of fruit in the salad, how many red grapes are in the salad?

# To answer this question, write a Python program to answer the following subquestions:

# 1. How many green grapes are there? (independent, support: [])

$green\_grapes = Symbol("green\_grapes")$

# 2. How many red grapes are there? (depends on 1, support: ["There are seven more than 3 times the number of red grapes as green grapes"])

$red\_grapes = 3 * green\_grapes + 7$

# 3. How many raspberries are there? (depends on 1, support: ["There are 5 less raspberries than green grapes"])

$raspberries = green\_grapes - 5$

# 4. How many total pieces of fruit are there? (depends on 1, 2, and 3, support: "there are 102 pieces of fruit in the salad")

$total\_fruit\_eq = Eq(green\_grapes + red\_grapes + raspberries, 102)$

# 5. How many green grapes are in the salad based on this equation? (depends on 1 and 4, support: [])

$green\_grapes\_val = solve\_it(total\_fruit\_eq, green\_grapes)[green\_grapes]$

# 6. How many red grapes are in the salad given the number of green grapes? (depends on 1, 2 and 5, support: "how many red grapes are in the salad?"])

$red\_grapes\_val = red\_grapes.subs(green\_grapes, green\_grapes\_val)$

# 7. Final Answer: how many red grapes are in the salad? (depends on 6, support: [])

$answer = red\_grapes\_val$

# Q: Denise will be 25 years old in two years. Her sister, Diane, is 4 years younger. In how many years will Diane be 25 years old?

# To answer this question, write a Python program to answer the following subquestions:

# 1. How old will Denise be in two years? (independent, support: ["Denise will be 25 years old in two years"])

$age\_denise\_in\_2y = 25$

# 2. How old is Denise now? (depends on 1, support: ["Denise will be 25 years old in two years"])

$age\_denise\_now = age\_denise\_in\_2y - 2$

# 3. How old is Diane now? (depends on 2, support: ["Diane, is 4 years younger"])

$age\_diane\_now = age\_denise\_now - 4$

# 4. In how many years will Diane be 25 years old? (depends on 3, support: [])

$years\_until\_diane\_25 = 25 - age\_diane\_now$

# 5. Final Answer: In how many years will Diane be 25 years old? (depends on 4, support: [])

$answer = years\_until\_diane\_25$

# Q: There were 90 people at the summer picnic. There were 50 soda cans, 50 plastic bottles of sparkling water, and 50 glass bottles of juice. One-half of the guests drank soda, one-third of the guests drank sparkling water, and four-fifths of the juices were consumed. How many recyclable cans and bottles were collected?

# To answer this question, write a Python program to answer the following subquestions:

# 1. How many people were at the summer picnic? (independent, support: ["There were 90 people"])

$people = 90$

# 2. How many soda cans were there? (independent, support: ["There were 50 soda cans"])

$soda\_cans = 50$

# 3. How many plastic bottles of sparkling water were there? (independent, support: ["50 plastic bottles of sparkling water"])

$sparkling\_water\_bottles = 50$

# 4. How many glass bottles of juice were there? (independent, support: ["50 glass bottles of juice"])

$juice\_bottles = 50$

# 5. How many guests drank soda? (depends on 1, support: ["One-half of the guests drank soda"])

$soda\_consumed = people * 1/2$

# 6. How many guests drank sparkling water? (depends on 1, support: ["one-third of the guests drank sparkling water"])

$sparkling\_water\_consumed = people * 1/3$

# 7. How many juices were consumed? (depends on 4, support: ["four-fifths of the juices were consumed"])

$juices\_consumed = juice\_bottles * 4/5$

# 8. How many recyclable cans and bottles were collected? (depends on 5, 6, and 7, support: [])

$cans\_and\_bottles\_collected = soda\_consumed + sparkling\_water\_consumed + juices\_consumed$

# 9. Final Answer: How many recyclable cans and bottles were collected? (depends on 8, support: [])

$answer = cans\_and\_bottles\_collected$

# Q: Mark has an egg farm. His farm supplies one store with 5 dozen eggs and another store with 30 eggs each day. How many eggs does he supply these two stores in a week?

# To answer this question, write a Python program to answer the following subquestions:

# 1. How many items are in one dozen? (independent, support: ["External knowledge: there are 12 items in a dozen"]

$n\_in\_dozen = 12$

# 2. How many eggs does Mark supply to one store each day? (depends on 1, support: ["supplies one store with 5 dozen eggs"])

$eggs\_one\_store\_each\_day = 5 * n\_in\_dozen$

# 3. How many eggs does Mark supply to another store each day? (independent, support: ["another store with 30 eggs each day"])

$eggs\_another\_store\_each\_day = 30$

# 4. How many days are there in a week? (independent, support: ["External knowledge: there are seven days in a week"])

$days\_in\_week = 7$

# 5. How many eggs does Mark supply these two stores in a week? (depends on 2, 3, and 4, support: [])

```
eggs_week = 0
for i in range(days_in_week):
    eggs_week += eggs_one_store_each_day + eggs_another_store_each_day
```

# 6. Final Answer: How many eggs does he supply these two stores in a week? (depends on 5, support: [])

$answer = eggs\_week$

# Q: Rebecca makes her own earrings out of buttons, magnets, and gemstones. For every earring, she uses two magnets, half as many buttons as magnets, and three times as many gemstones as buttons. If Rebecca wants to make 4 sets of earrings, how many gemstones will she need?

# To answer this question, write a Python program to answer the following subquestions:

# 1. How many magnets does Rebecca need for one earring? (independent, support: ["For every earring, she uses two magnets"])

$magnets\_per\_earring = 2$

# 2. How many earrings are in a set? (independent, support: ["External knowledge: there are two earrings in a set, one per ear"])

$earrings\_per\_set = 2$

# 3. How many magnets does Rebecca need for one set of earrings? (depends on 1 and 2, support: [])

$magnets\_per\_set = magnets\_per\_earring * earrings\_per\_set$

# 4. How many buttons does Rebecca need for one set of earrings? (depends on 3, support: ["half as many buttons as magnets"])

$buttons\_per\_set = magnets\_per\_set * 1/2$

# 5. How many gemstones does Rebecca need for one set of earrings? (depends on 4, support: ["three times as many gemstones as buttons"])

$gemstones\_per\_set = buttons\_per\_set * 3$

# 6. How many gemstones does Rebecca need for 4 sets of earrings? (depends on 5, support: ["Rebecca wants to make 4 sets of earrings"])

$gemstones\_4\_sets = gemstones\_per\_set * 4$

# 7. Final Answer: If Rebecca wants to make 4 sets of earrings, how many gemstones will she need? (depends on 6, support: [])

$answer = gemstones\_4\_sets$

# Follow the format of the examples above to answer the following question. Don't say anything else at the beginning or at the end of your answer. Just give the code with the comments as in the examples.

# Q: [QUESTION]

# To answer this question, write a Python program to answer the following subquestions:"

8.1.3 Game of 24

Standard IO Prompting

System Prompt

"Give the final answer in the way shown in the examples, like 'Answer: your_answer'."

User Prompt (corresponding number of examples based on the shot count)

"Use numbers and basic arithmetic operations (+ - * /) to obtain 24.

Input: 4 4 6 8

Answer: (4 + 8) * (6 - 4) = 24

Input: 2 9 10 12

Answer: 2 * 12 * (10 - 9) = 24

Input: 4 9 10 13

Answer: (13 - 9) * (10 - 4) = 24

Input: 1 4 8 8

Answer: (8 / 4 + 1) * 8 = 24

Input: 5 5 5 9

Answer: 5 + 5 + 5 + 9 = 24

Input: 1 5 5 9

Answer: (1 + 5) * (9 - 5) = 24

Input: 5 10 12 13

Answer: (5 + 10 - 13) * 12 = 24

Input: 4 5 8 13

Answer: 4 * 8 + 5 - 13 = 24

Input: state

Answer:"

Chain-of-Thought (CoT) Prompting

System Prompt

"Follow a step-by-step solution as shown in the examples and give your final answer as: 'Answer: your_answer'."

User Prompt (corresponding number of examples based on the shot count)

"Use the input numbers and basic arithmetic operations (+ - * /) to obtain 24. Each step, you are only allowed to choose two of the remaining numbers to obtain a new number.

Input: 4 4 6 8

Steps:

4 + 8 = 12 (left: 4 6 12)

6 - 4 = 2 (left: 2 12)

2 * 12 = 24 (left: 24)

Answer: (6 - 4) * (4 + 8) = 24

Input: 2 9 10 12

Steps: 12 * 2 = 24 (left: 9 10 24)

10 - 9 = 1 (left: 1 24)

24 * 1 = 24 (left: 24)

Answer: (12 * 2) * (10 - 9) = 24

Input: 4 9 10 13

Steps:

13 - 10 = 3 (left: 3 4 9)

9 - 3 = 6 (left: 4 6)

4 * 6 = 24 (left: 24)

Answer: 4 * (9 - (13 - 10)) = 24

Input: 1 4 8 8

Steps:

8 / 4 = 2 (left: 1 2 8)

1 + 2 = 3 (left: 3 8)

3 * 8 = 24 (left: 24)

Answer: (1 + 8 / 4) * 8 = 24

Input: 5 5 5 9

Steps:

5 + 5 = 10 (left: 5 9 10)

10 + 5 = 15 (left: 9 15)

15 + 9 = 24 (left: 24)

Answer: ((5 + 5) + 5) + 9 = 24

Input: 1 5 5 9

Steps:

1 + 5 = 6 (left: 5 6 9)

9 - 5 = 4 (left: 4 6)

4 * 6 = 24 (left: 24)

Answer: (1 + 5) * (9 - 5) = 24

Input: 5 10 12 13

Steps:

5 + 10 = 15 (left: 12 13 15)

15 - 13 = 2 (left: 2 12)

2 * 12 = 24 (left: 24)

Answer: (5 + 10 - 13) * 12 = 24

Input: 4 5 8 13

Steps:

4 * 8 = 32 (left: 5 13 32)

5 + 32 = 37 (left: 13 37)

37 - 13 = 24 (left: 24)

Answer: 4 * 8 + 5 - 13 = 24

Input: state

Answer:"

Puzzle Translation

System Prompt

You are an expert assistant specializing in solving problems by writing Python programs. Ensure your response can be directly executed as a Python script. Any non-code explanations should be included as comments, and the Python code should be enclosed within triple backticks (“‘python). Finally, store your complete solution in a variable called ’answer’.

User Prompt (Indicating the steps of reasoning)

"Use the input numbers and basic arithmetic operations (+ - * /) to obtain 24. Your answer should be an expression with numbers and operations. Do NOT print the answer, just store it.

Input: state

Generate the required code for the steps below:

# 1. What are the available numbers?

# 2. What is the target result?

# 3. What operations can be used between the numbers?

# 4. Explore all possible permutations of the numbers

# 5. Explore all possible combinations of the three operations

# 6. Different possible ways to group numbers and operations

# 7. Evaluate each expression to see if it equals the target result

# 8. Store the solution in the ’answer’ variable

Answer:"

Chapter 9

Bibliography

[1] Agostinelli, F. et al. "Solving the Rubik's cube with deep reinforcement learning and search". In: *Nature Machine Intelligence* 1 (2019), pp. 356–363. URL:

[2] Anil, R. et al. "PaLM 2 Technical Report". In: *ArXiv* abs/2305.10403 (2023). URL:

[3] Argyrou, G. et al. *Automatic Generation of Fashion Images using Prompting in Generative Machine Learning Models*. 2024. arXiv: 2407.14944 [cs.CV]. URL:

[4] Bang, Y. et al. "A Multitask, Multilingual, Multimodal Evaluation of ChatGPT on Reasoning, Hallucination, and Interactivity". In: *ArXiv* abs/2302.04023 (2023). URL:

[5] Bao, Q. et al. "A Systematic Evaluation of Large Language Models on Out-of-Distribution Logical Reasoning Tasks". In: *ArXiv* abs/2310.09430 (2023). URL:

[6] Barj, H. N. E. and Sautory, T. *Reinforcement Learning from LLM Feedback to Counteract Goal Misgeneralization*. 2024. arXiv: 2401.07181 [cs.LG].

[7] Besta, M. et al. *Graph of Thoughts: Solving Elaborate Problems with Large Language Models*. 2023. arXiv: 2308.09687 [cs.CL].

[8] Besta, M. et al. *Demystifying Chains, Trees, and Graphs of Thoughts*. 2024. arXiv: 2401.14295 [cs.CL].

[9] Bisk, Y. et al. "PIQA: Reasoning about Physical Commonsense in Natural Language". In: *ArXiv* abs/1911.11641 (2019). URL:

[10] Brown, T. B. et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL]. URL:

[11] Brown, T. B. et al. "Language Models are Few-Shot Learners". In: *ArXiv* abs/2005.14165 (2020). URL:

[12] Campbell, M., Hoane, A., and Hsu, F.-h. "Deep Blue". In: *Artificial Intelligence* 134.1 (2002), pp. 57–83. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(01)00129-1. URL:

[13] Chen, B. et al. "Unleashing the potential of prompt engineering in Large Language Models: a comprehensive review". In: *ArXiv* abs/2310.14735 (2023). URL:

[14] Chen, J. "Different Algorithms to Solve a Rubik's Cube". In: *Journal of Physics: Conference Series* 2386.1 (Dec. 2022), p. 012018. DOI: 10.1088/1742-6596/2386/1/012018. URL:

[15] Chen, M. et al. "Evaluating Large Language Models Trained on Code". In: *ArXiv* abs/2107.03374 (2021). URL:

[16] Chen, W. et al. "Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks". In: *Trans. Mach. Learn. Res.* 2023 (2022). URL:

[17] Chi, E. C. and Lange, K. *Techniques for Solving Sudoku Puzzles*. 2013. arXiv: 1203.2295 [math.OC].

[18] Chu, Z. et al. "A Survey of Chain of Thought Reasoning: Advances, Frontiers and Future". In: *ArXiv* abs/2309.15402 (2023). URL:

[19] Chung, H. W. et al. "Scaling Instruction-Finetuned Language Models". In: *ArXiv* abs/2210.11416 (2022). URL:

[20] Cobbe, K. et al. "Training Verifiers to Solve Math Word Problems". In: *ArXiv* abs/2110.14168 (2021). URL:

[21] Conneau, A. et al. "Unsupervised Cross-lingual Representation Learning at Scale". In: *Annual Meeting of the Association for Computational Linguistics*. 2019. URL:

[22] Creswell, A., Shanahan, M., and Higgins, I. “Selection-Inference: Exploiting Large Language Models for Interpretable Logical Reasoning”. In: *ArXiv* abs/2205.09712 (2022). URL:

[23] Del, M. and Fishel, M. “True Detective: A Deep Abductive Reasoning Benchmark Undoable for GPT-3 and Challenging for GPT-4”. In: *STARSEM*. 2022. URL:

[24] Devlin, J. et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL]. URL:

[25] Ding, R. et al. “Everything of Thoughts: Defying the Law of Penrose Triangle for Thought Generation”. In: *ArXiv* abs/2311.04254 (2023). URL:

[26] Dong, Q. et al. *A Survey on In-context Learning*. 2023. arXiv: 2301.00234 [cs.CL].

[27] Dubey, A. et al. *The Llama 3 Herd of Models*. 2024. arXiv: 2407.21783 [cs.AI]. URL:

[28] Efrat, A. et al. “Cryptonite: A Cryptic Crossword Benchmark for Extreme Ambiguity in Language”. In: *ArXiv* abs/2103.01242 (2021). URL:

[29] Feng, J. et al. “Language Models can be Logical Solvers”. In: *ArXiv* abs/2311.06158 (2023). URL:

[30] Feng, X. et al. “ChessGPT: Bridging Policy Learning and Language Modeling”. In: *ArXiv* abs/2306.09200 (2023). URL:

[31] Filandrianos, G. et al. *Counterfactuals of Counterfactuals: a back-translation-inspired approach to analyse counterfactual editors*. 2023. arXiv: 2305.17055 [cs.CL]. URL:

[32] Flach, P. A. and Kakas, A. C. “Abductive and inductive reasoning: background and issues”. In: 2000. URL:

[33] Fu, Y. et al. “Complexity-Based Prompting for Multi-Step Reasoning”. In: *ArXiv* abs/2210.00720 (2022). URL:

[34] Gage, P. “A new algorithm for data compression”. In: *The C Users Journal archive* 12 (1994), pp. 23–38. URL:

[35] Gao, L. et al. “PAL: Program-aided Language Models”. In: *ArXiv* abs/2211.10435 (2022). URL:

[36] Geva, M. et al. “Did Aristotle Use a Laptop? A Question Answering Benchmark with Implicit Reasoning Strategies”. In: *Transactions of the Association for Computational Linguistics* 9 (2021), pp. 346–361. URL:

[37] Ghosal, D. et al. “Are Language Models Puzzle Prodigies? Algorithmic Puzzles Unveil Serious Challenges in Multimodal Reasoning”. In: *ArXiv* abs/2403.03864 (2024). URL:

[38] Giadikiaroglou, P. et al. *Puzzle Solving using Reasoning of Large Language Models: A Survey*. 2024. arXiv: 2402.11291 [cs.CL]. URL:

[39] Grigoriadou, N. et al. *AILS-NTUA at SemEval-2024 Task 6: Efficient model tuning for hallucination detection and analysis*. 2024. arXiv: 2404.01210 [cs.CL]. URL:

[40] Gu, Z. et al. “Go Beyond The Obvious: Probing the gap of INFORMAL reasoning ability between Humanity and LLMs by Detective Reasoning Puzzle Benchmark”. In: 2023. URL:

[41] Guo, J. et al. “Suspicion-Agent: Playing Imperfect Information Games with Theory of Mind Aware GPT-4”. In: *ArXiv* abs/2309.17277 (2023). URL:

[42] Gupta, A. “Are ChatGPT and GPT-4 Good Poker Players? - A Pre-Flop Analysis”. In: *ArXiv* abs/2308.12466 (2023). URL:

[43] Hochreiter, S. and Schmidhuber, J. “Long Short-Term Memory”. In: *Neural Computation* 9 (1997), pp. 1735–1780. URL:

[44] Huang, C. et al. “PokerGPT: An End-to-End Lightweight Solver for Multi-Player Texas Hold’em via Large Language Model”. In: *ArXiv* abs/2401.06781 (2024). URL:

[45] Huang, J. and Chang, K. C.-C. “Towards Reasoning in Large Language Models: A Survey”. In: *ArXiv* abs/2212.10403 (2022). URL:

[46] Huang, L. et al. *A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions*. 2023. arXiv: 2311.05232 [cs.CL]. URL:

[47] Huang, S. et al. “LatEval: An Interactive LLMs Evaluation Benchmark with Incomplete Information from Lateral Thinking Puzzles”. In: *ArXiv* abs/2308.10855 (2023). URL:

[48] Ishay, A., Yang, Z., and Lee, J. “Leveraging Large Language Models to Generate Answer Set Programs”. In: *ArXiv* abs/2307.07699 (2023). URL:

[49] Jiang, A. Q. et al. “Mistral 7B”. In: *ArXiv* abs/2310.06825 (2023). URL:

[50] Jiang, Y., Ilievski, F., and Ma, K. “BRAINTEASER: Lateral Thinking Puzzles for Large Language Models”. In: *Conference on Empirical Methods in Natural Language Processing*. 2023. URL:

[51] Kazemi, M. et al. "BoardgameQA: A Dataset for Natural Language Reasoning with Contradictory Information". In: *ArXiv* abs/2306.07934 (2023). URL:

[52] Khashabi, D. et al. "UnifiedQA: Crossing Format Boundaries With a Single QA System". In: *Findings*. 2020. URL:

[53] Kojima, T. et al. *Large Language Models are Zero-Shot Reasoners*. 2023. arXiv: 2205.11916 [cs.CL]. URL:

[54] Koncel-Kedziorski, R. et al. "MAWPS: A Math Word Problem Repository". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Ed. by K. Knight, A. Nenkova, and O. Rambow. San Diego, California: Association for Computational Linguistics, June 2016, pp. 1152–1157. DOI: 10.18653/v1/N16-1136. URL:

[55] Korf, R. E. "Finding Optimal Solutions to Rubik's Cube Using Pattern Databases". In: *AAAI/IAAI*. 1997. URL:

[56] Koulakos, A. et al. *Enhancing adversarial robustness in Natural Language Inference using explanations*. 2024. arXiv: 2409.07423 [cs.CL]. URL:

[57] Kritharoula, A., Lymperaiou, M., and Stamou, G. *Language Models as Knowledge Bases for Visual Word Sense Disambiguation*. 2023. arXiv: 2310.01960 [cs.CL]. URL:

[58] Kritharoula, A., Lymperaiou, M., and Stamou, G. "Large Language Models and Multimodal Retrieval for Visual Word Sense Disambiguation". In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by H. Bouamor, J. Pino, and K. Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 13053–13077. DOI: 10.18653/v1/2023.emnlp-main.807. URL:

[59] Kulshreshtha, S. et al. "Down and Across: Introducing Crossword-Solving as a New NLP Benchmark". In: *ArXiv* abs/2205.10442 (2022). URL:

[60] Lan, Y. et al. "LLM-Based Agent Society Investigation: Collaboration and Confrontation in Avalon Gameplay". In: *ArXiv* abs/2310.14985 (2023). URL:

[61] Lan, Z. et al. "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations". In: *ArXiv* abs/1909.11942 (2019). URL:

[62] Lei, B. et al. "Boosting Logical Reasoning in Large Language Models through a New Framework: The Graph of Thought". In: *ArXiv* abs/2308.08614 (2023). URL:

[63] Lester, B., Al-Rfou, R., and Constant, N. "The Power of Scale for Parameter-Efficient Prompt Tuning". In: *Conference on Empirical Methods in Natural Language Processing*. 2021. URL:

[64] Lewis, M. et al. "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension". In: *Annual Meeting of the Association for Computational Linguistics*. 2019. URL:

[65] Lewis, M. et al. *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. 2019. arXiv: 1910.13461 [cs.CL]. URL:

[66] Li, Y., Wang, H., and Zhang, C. "Assessing Logical Puzzle Solving in Large Language Models: Insights from a Minesweeper Case Study". In: *ArXiv* abs/2311.07387 (2023). URL:

[67] Lin, B. Y. et al. "RiddleSense: Reasoning about Riddle Questions Featuring Linguistic Creativity and Commonsense Knowledge". In: *Findings*. 2021. URL:

[68] Lin, B. Y. et al. *RiddleSense: Reasoning about Riddle Questions Featuring Linguistic Creativity and Commonsense Knowledge*. 2021. arXiv: 2101.00376 [cs.CL]. URL:

[69] Liu, H. et al. "Evaluating the Logical Reasoning Ability of ChatGPT and GPT-4". In: *ArXiv* abs/2304.03439 (2023). URL:

[70] Liu, H. et al. "GLoRE: Evaluating Logical Reasoning of Large Language Models". In: *ArXiv* abs/2310.09107 (2023). URL:

[71] Liu, P. et al. "Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing". In: *ACM Computing Surveys* 55 (2021), pp. 1–35. URL:

[72] Liu, W. et al. "Mathematical Language Models: A Survey". In: *ArXiv* abs/2312.07622 (2023). URL:

[73] Liu, Y. et al. "RoBERTa: A Robustly Optimized BERT Pretraining Approach". In: *ArXiv* abs/1907.11692 (2019). URL:

[74] Long, J. "Large Language Model Guided Tree-of-Thought". In: *ArXiv* abs/2305.08291 (2023). URL:

[75] Luo, M. et al. "Towards LogiGLUE: A Brief Survey and A Benchmark for Analyzing Logical Reasoning Capabilities of Language Models". In: *ArXiv* abs/2310.00836 (2023). URL:

[76] Luo, M. et al. *Towards LogiGLUE: A Brief Survey and A Benchmark for Analyzing Logical Reasoning Capabilities of Language Models*. 2024. arXiv: 2310.00836 [cs.CL]. URL:

[77] Lymperaiou, M. et al. *Towards explainable evaluation of language models on the semantic similarity of visual concepts*. 2022. arXiv: 2209.03723 [cs.CL]. URL:

[78] Lyu, Q. et al. "Faithful Chain-of-Thought Reasoning". In: *ArXiv* abs/2301.13379 (2023). URL:

[79] Madaan, A. et al. "Self-Refine: Iterative Refinement with Self-Feedback". In: *ArXiv* abs/2303.17651 (2023). URL:

[80] Markaki, S. and Panagiotakis, C. "Jigsaw puzzle solving techniques and applications: a survey". In: *The Visual Computer* 39 (2022), pp. 4405–4421. URL:

[81] McAleer, S. et al. *Solving the Rubik's Cube Without Human Knowledge*. 2018. arXiv: 1805.07470 [cs.AI].

[82] Miao, S.-Y., Liang, C.-C., and Su, K.-Y. "A Diverse Corpus for Evaluating and Developing English Math Word Problem Solvers". In: *Annual Meeting of the Association for Computational Linguistics*. 2020. URL:

[83] Mikolov, T. et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL]. URL:

[84] Minaee, S. et al. *Large Language Models: A Survey*. 2024. arXiv: 2402.06196 [cs.CL]. URL:

[85] Mitra, A. and Baral, C. "Learning to Automatically Solve Logic Grid Puzzles". In: *Conference on Empirical Methods in Natural Language Processing*. 2015. URL:

[86] Mo, S. and Xin, M. "Tree of Uncertain Thoughts Reasoning for Large Language Models". In: *ArXiv* abs/2309.07694 (2023). URL:

[87] Noever, D. A. and Burdick, R. "Puzzle Solving without Search or Human Knowledge: An Unnatural Language Approach". In: *ArXiv* abs/2109.02797 (2021). URL:

[88] Olausson, T. X. et al. "LINC: A Neurosymbolic Approach for Logical Reasoning by Combining Language Models with First-Order Logic Provers". In: *Conference on Empirical Methods in Natural Language Processing*. 2023. URL:

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

[90] Pan, L. et al. "Logic-LM: Empowering Large Language Models with Symbolic Solvers for Faithful Logical Reasoning". In: *ArXiv* abs/2305.12295 (2023). URL:

[91] Panagiotopoulos, I. et al. "AILS-NTUA at SemEval-2024 Task 9: Cracking Brain Teasers: Transformer Models for Lateral Thinking Puzzles". In: *ArXiv* abs/2404.01084 (2024). URL:

[92] Panagiotopoulos, I. et al. *RISCORE: Enhancing In-Context Riddle Solving in Language Models through Context-Reconstructed Example Augmentation*. 2024. arXiv: 2409.16383 [cs.CL]. URL:

[93] Papadimitriou, C. et al. *Masked Generative Story Transformer with Character Guidance and Caption Augmentation*. 2024. arXiv: 2403.08502 [cs.CV]. URL:

[94] Patel, A., Bhattamishra, S., and Goyal, N. "Are NLP Models really able to Solve Simple Math Word Problems?" In: *North American Chapter of the Association for Computational Linguistics*. 2021. URL:

[95] Pennington, J., Socher, R., and Manning, C. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by A. Moschitti, B. Pang, and W. Daelemans. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL:

[96] Peters, M. E. et al. *Deep contextualized word representations*. 2018. arXiv: 1802.05365 [cs.CL]. URL:

[97] Pourcel, J. et al. "ACES: Generating Diverse Programming Puzzles with Autotelic Language Models and Semantic Descriptors". In: *ArXiv* abs/2310.10692 (2023). URL:

[98] Qiao, S. et al. "Reasoning with Language Model Prompting: A Survey". In: *ArXiv* abs/2212.09597 (2022). URL:

[99] Radford, A. and Narasimhan, K. "Improving Language Understanding by Generative Pre-Training". In: 2018. URL:

[100] Radford, A. et al. "Language Models are Unsupervised Multitask Learners". In: 2019. URL:

[101] Raffel, C. et al. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: *J. Mach. Learn. Res.* 21 (2019), 140:1–140:67. URL:

[102] Raffel, C. et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2023. arXiv: 1910.10683 [cs.LG]. URL:

[103] Rosenblatt, F. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65 6 (1958), pp. 386–408. URL:

[104] Rozner, J., Potts, C., and Mahowald, K. “Decrypting Cryptic Crosswords: Semantically Complex Wordplay Puzzles as a Target for NLP”. In: *ArXiv* abs/2104.08620 (2021). URL:

[105] Saparov, A. et al. “Testing the General Deductive Reasoning Capacity of Large Language Models Using OOD Examples”. In: *ArXiv* abs/2305.15269 (2023). URL:

[106] Savelka, J. et al. *Large Language Models (GPT) Struggle to Answer Multiple-Choice Questions about Code*. 2023. arXiv: 2303.08033 [cs.CL].

[107] Schuster, T. et al. “Programming Puzzles”. In: *ArXiv* abs/2106.05784 (2021). URL:

[108] Sherstinsky, A. “Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network”. In: *Physica D: Nonlinear Phenomena* 404 (Mar. 2020), p. 132306. ISSN: 0167-2789. DOI: 10.1016/j.physd.2019.132306. URL:

[109] Silver, D. et al. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. 2017. arXiv: 1712.01815 [cs.AI].

[110] Simonis, H. “Sudoku as a Constraint Problem”. In: 2005. URL:

[111] Studholme, C. “Minesweeper as a Constraint Satisfaction Problem”. In: 2001. URL:

[112] Szomiu, R. and Groza, A. “A Puzzle-Based Dataset for Natural Language Inference”. In: *ArXiv* abs/2112.05742 (2021). URL:

[113] Takano, K. *Self-Supervision is All You Need for Solving Rubik’s Cube*. 2023. arXiv: 2106.03157 [cs.LG].

[114] Talmor, A. et al. “CommonsenseQA: A Question Answering Challenge Targeting Commonsense Knowledge”. In: *ArXiv* abs/1811.00937 (2019). URL:

[115] Thomas, K. et al. *"I Never Said That": A dataset, taxonomy and baselines on response clarity classification*. 2024. arXiv: 2409.13879 [cs.CL]. URL:

[116] Tong, Y. et al. “Eliminating Reasoning via Inferring with Planning: A New Framework to Guide LLMs’ Non-linear Thinking”. In: *ArXiv* abs/2310.12342 (2023). URL:

[117] Touvron, H. et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: 2307.09288 [cs.CL]. URL:

[118] Touvron, H. et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL]. URL:

[119] Vaswani, A. et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL:

[120] Wang, L. et al. “Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models”. In: *Annual Meeting of the Association for Computational Linguistics*. 2023. URL:

[121] Wang, S. et al. “Avalon’s Game of Thoughts: Battle Against Deception through Recursive Contemplation”. In: *ArXiv* abs/2310.01320 (2023). URL:

[122] Wang, W. et al. “CAR: Conceptualization-Augmented Reasoner for Zero-Shot Commonsense Question Answering”. In: *Conference on Empirical Methods in Natural Language Processing*. 2023. URL:

[123] Wang, X. et al. “Self-Consistency Improves Chain of Thought Reasoning in Language Models”. In: *ArXiv* abs/2203.11171 (2022). URL:

[124] Wang, X. et al. *Self-Consistency Improves Chain of Thought Reasoning in Language Models*. 2023. arXiv: 2203.11171 [cs.CL]. URL:

[125] Wei, J. et al. “Chain of Thought Prompting Elicits Reasoning in Large Language Models”. In: *Advances in Neural Information Processing Systems*. Ed. by A. H. Oh et al. 2022. URL:

[126] Wei, J. et al. “Emergent Abilities of Large Language Models”. In: *ArXiv* abs/2206.07682 (2022). URL:

[127] Wei, J. et al. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. 2023. arXiv: 2201.11903 [cs.CL]. URL:

[128] Xu, F., Zhang, Y., and Wan, X.-Y. “CC-Riddle: A Question Answering Dataset of Chinese Character Riddles”. In: *ArXiv* abs/2206.13778 (2022). URL:

[129] Xu, F. et al. *Are Large Language Models Really Good Logical Reasoners? A Comprehensive Evaluation and Beyond*. 2023. arXiv: 2306.09841 [cs.CL].

[130] Xu, F. et al. “Are Large Language Models Really Good Logical Reasoners? A Comprehensive Evaluation and Beyond”. In: 2023. URL:

[131] Xu, Y. et al. “Exploring Large Language Models for Communication Games: An Empirical Study on Werewolf”. In: *ArXiv* abs/2309.04658 (2023). URL:

[132] Yang, S. et al. “Neuro-Symbolic Integration Brings Causal and Reliable Reasoning Proofs”. In: *ArXiv* abs/2311.09802 (2023). URL:

[133] Yang, Z. et al. *Logical Reasoning over Natural Language as Knowledge Representation: A Survey*. 2024. arXiv: 2303.12023 [cs.CL]. URL:

[134] Yao, S. et al. "Tree of Thoughts: Deliberate Problem Solving with Large Language Models". In: *ArXiv* abs/2305.10601 (2023). URL:

[135] Yu, F. et al. *Natural Language Reasoning, A Survey*. 2023. arXiv: 2303.14725 [cs.CL]. URL:

[136] Yu, Z. et al. "Towards Better Chain-of-Thought Prompting Strategies: A Survey". In: *ArXiv* abs/2310.04959 (2023). URL:

[137] Zeinalipour, K. et al. "ArabIcros: AI-Powered Arabic Crossword Puzzle Generation for Educational Applications". In: *Proceedings of ArabicNLP 2023*. Association for Computational Linguistics, 2023. DOI: 10.18653/v1/2023.arabicnlp-1.23. URL:

[138] Zeinalipour, K. et al. *Italian Crossword Generator: Enhancing Education through Interactive Word Puzzles*. 2023. arXiv: 2311.15723 [cs.CL].

[139] Zhang, Y. and Wan, X. "BiRdQA: A Bilingual Dataset for Question Answering on Tricky Riddles". In: *ArXiv* abs/2109.11087 (2021). URL:

[140] Zhang, Z. et al. "Automatic Chain of Thought Prompting in Large Language Models". In: *ArXiv* abs/2210.03493 (2022). URL:

[141] Zhao, H. et al. *Explainability for Large Language Models: A Survey*. 2023. arXiv: 2309.01029 [cs.CL]. URL:

[142] Zhao, J. and Anderson, C. J. *Solving and Generating NPR Sunday Puzzles with Large Language Models*. 2023. arXiv: 2306.12255 [cs.CL].

[143] Zhou, D. et al. "Least-to-Most Prompting Enables Complex Reasoning in Large Language Models". In: *ArXiv* abs/2205.10625 (2022). URL:

[144] Zhou, Y. et al. "Large Language Models Are Human-Level Prompt Engineers". In: *ArXiv* abs/2211.01910 (2022). URL:

[145] Zugarini, A. et al. *Clue-Instruct: Text-Based Clue Generation for Educational Crossword Puzzles*. 2024. arXiv: 2404.06186 [cs.CL].