Indian Institute of Technology Madras

Department of Data Science and Artificial Intelligence

Dual Degree Project Report

Project Title:

Exploring Reasoning Abilities of Large Language Models

Author:

Pranoy K P

ED20B045

IDDD Data Science

Guides:

Dr. Sivaram Ambikasaran, Dr. Sri Vallabha Deevi

Department of Data Science and AI

Contents

Abstract

Large Language Models (LLMs) have revolutionised natural language processing by demonstrating remarkable language understanding, generation, and task generalisation capabilities. LLMs are language models with many parameters, trained with self-supervised learning on a vast amount of text. Modern models can be fine-tuned for specific tasks or guided by prompt engineering. However, their ability to reason and plan remains an open research challenge. Using a structured evaluation across various tasks, this project investigates LLMs' reasoning abilities, such as Gemini Flash, Llama, OpenHermes, DeepSeek, Gemma, etc. We explore methods like Chain-of-Thought (CoT) prompting, Retrieval-Augmented Generation (RAG), and Program-Aided Language Models (PAL), both in static and dynamic prompting setups. Our methodology involves benchmarking baseline performance and measuring the improvements introduced through these techniques. We survey the capabilities of AI agents (programs that can autonomously perform a task on behalf of a user) to handle reasoning tasks. We also assess the consistency of reasoning with and without internet access, evaluate the impact of model size on performance, and compare open-source and closed-source LLMs in terms of cost, accessibility, and accuracy. We have also verified whether closed-source models outperform open models in complex reasoning tasks. We also survey whether prompting techniques and support methodologies can narrow this gap. Dynamic few-shot prompting and PAL strategies significantly improve mathematical and multi-step reasoning task performance. The findings suggest that LLMs serve best as reasoning aids when augmented with external tools rather than standalone reasoning engines. This project helps to understand the practical limitations of LLMs in reasoning contexts and aids in building more interpretable and reliable LLM-powered systems.

Keywords: Large Language Models, Reasoning, Chain-of-Thought Prompting, Retrieval-Augmented Generation, Program-Aided Language Models, Agentic Frameworks

GitHub Link: https://github.com/juicebocks27/DDP-Reasoning

Thesis Certificate

This is to certify that the thesis titled **"Exploring Reasoning Abilities of Large Language Models"**, submitted by **Pranoy K P (ED20B045)** in partial fulfillment of the requirements for the award of the degree of **Master of Technology in Data Science**, is a bona fide record of the research work carried out by him during the academic year **2024–2025** in the **Department of Data Science, IIT Madras, India**, under our supervision. The contents of this thesis, in full or in part, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Dr. Sivaram Ambikasaran
Associate Professor,
Department of Mathematics,
Department of Data Science & Artificial Intelligence,
Wadhwani School of Data Science & Artificial Intelligence,
Indian Institute of Technology Madras

Dr. Sri Vallabha Deevi
Director, Data Science, Tiger Analytics
Adjunct Faculty, Department of Data Science & Artificial Intelligence, IIT Madras
Chennai, India

Date: May 10, 2025

Acknowledgements

I would like to express my sincere gratitude to Prof. Sivaram Ambikasaran and Dr. Sri Vallabha Deevi for their constant help and support, without which the completion of this project would not have been possible. Their insightful guidance and patient mentorship throughout the academic year have been invaluable.

I would also like to thank the professors of IIT Madras, whose teaching and encouragement laid the foundation for this work. A heartfelt thanks to my friends for their unwavering support, encouragement, and the many insightful discussions that helped me navigate various challenges during this journey. I am also thankful to my family for their unconditional love and constant encouragement, which have been my greatest source of strength throughout this academic pursuit.

Finally, I am grateful to the Department of Data Science & Artificial Intelligence, IIT Madras, and Tiger Analytics for providing me the opportunity to contribute to this exciting and rapidly evolving domain through a highly engaging and meaningful study.

Chapter 1

Introduction

Background

Reasoning abilities in Large Language Models (LLMs) are one of the key utilities and have wide-reaching applications, from answering complex questions and solving puzzles to coding and conversational talks. It may also open pathways to Artificial General Intelligence (AGI), where we may see AI replicating a human's cognitive abilities to complete all the tasks a human can perform. Some of these abilities include finding logical pathways based on provided information, making inferences, and arriving at accurate conclusions, which currently prove a challenge for the existing AI architectures. Despite their impressive capability to generate coherent, fluent texts LLMs often struggle with complex reasoning tasks. This is because they are not equipped to handle systematic thinking, logical deduction, and multi-step problem solving like a human brain does[1].

Enhancing these reasoning capabilities is critical to improving the reliability, accuracy, and real-world utility of LLMs, especially in scenarios requiring robust logical inference.

Recent advances in reasoning enhancement techniques that aim to bridge this gap are:

- **Chain-of-Thought (CoT) Prompting:** Forces models to decompose problems into intermediate steps, improving accuracy on math and logic tasks by around 10%[2].
- **Program-Aided Language Models (PAL):** Offload computation to external interpreters (e.g., Python), ensuring mathematically sound solutions while bypassing LLMs' arithmetic weaknesses.
- **Retrieval-Augmented Generation (RAG):** Enhances reasoning by retrieving relevant external knowledge chunks, grounding the model's responses in factual context.
- **Reinforcement Learning (RL) Integration:** Optimises reasoning pathways through reward mechanisms, enabling adaptive exploration of solution spaces.

Despite these innovations, the benchmarks reveal persistent limitations. For example, GPT-4's accuracy drops from 30% to near-zero on obfuscated planning tasks, exposing its reliance on lexical pattern matching over true reasoning[3]. Similarly, PAL models struggle with non-algorithmic problems requiring common-sense inference. RAG methodologies also present challenges, as they may introduce irrelevant context, adding unnecessary tokens to the LLMs and diverting attention from the core problem.

Problem Statement

Traditionally, reasoning challenges in LLMs have been addressed through prompt engineering, fine-tuning, and distillation of models. While these methods offer some performance gains, they often fail to generalise across problem types and domains. Furthermore, increasing the model's size by scaling from billions to trillions of parameters has not proven sufficient to overcome the core limitations of logical reasoning[4]. Besides, in high-level applications, inconsistent performance and lack of explainability lead to difficulty in adoption. There is a growing need to identify techniques that offer improved accuracy and transparency in reasoning pathways, particularly in tasks requiring structured thought and inferential depth.

Aims

This project explores and benchmarks the reasoning capabilities of LLMs across various tasks and reasoning aid techniques. We evaluate models such as Gemini, Llama and DeepSeek, spanning multiple parameter scales (e.g., 8B, 70B), to study how architecture and scale influence performance. We implement and compare baseline responses with advanced reasoning techniques, including Program-Aided Language models using Python and Resource Description Framework[5] code, static and dynamic few-shot prompting, and Agentic Chain-of-Thought (CoT) with and without internet access. The study's objective is to analyse each approach regarding accuracy, reasoning coherence, and generalisation across problem types to identify methodologies that most effectively enhance LLMs' structured reasoning abilities to obtain more nuanced, reliable solutions beyond surface-level information retrieval.

This project conducts a comprehensive analysis of LLM reasoning capabilities, focusing on three axes of improvement.

- **Methodology Benchmarking:**
  - Comparing Retrieval-Augmented Generation with and without reasoning cues
  - Contrast static one-shot prompting vs. dynamic few-shot adaptation
  - Compare PAL for code generation with plain Python and code generation with Reasoning Engine Frameworks
  - Compare Agentic CoT Prompting with and without internet access
- **Architectural Comparisons:**
  - Evaluate open-source models (Llama) against closed-source systems (Gemini)
  - Test parameter scaling effects using 8B and 70B variants
  - Compare the performance of models with specialised training, like conversational, coding, and reasoning models
- **Generalisation Assessment:**
  - Measure performance drift while repeating experiments across multiple runs

Chapter 2

Literature Review

This chapter reviews current research on reasoning and planning capabilities in Large Language Models. We examine the fundamental capabilities of LLMs for logical reasoning, their limitations in solving complex reasoning tasks, and the methodologies developed to enhance these capabilities.

2.1 Can LLMs Reason and Plan

LLMs are not inherently designed for principled reasoning, as evidenced by their training methodology and primary use cases. True reasoning often requires computationally hard inference processes, whereas LLMs excel primarily at "approximate retrieval," probabilistically reconstructing responses word by word based on patterns in their training data. Although this approach enables creativity and flexibility in responses, it also leads to the well-documented phenomenon of hallucinations, as LLMs fundamentally struggle with strict memorisation and verification of facts.

Karthik Valmeekam et al. [6] evaluated GPT-3's planning abilities using standardised tasks from the International Planning Competition, including benchmark problems such as Blocks World. Their results contradicted anecdotal claims about LLMs' reasoning capabilities, revealing severely limited performance in structured planning tasks. Subsequent testing on more advanced models like GPT-3.5 and GPT-4 showed modest improvements, with GPT-4 achieving approximately 30% accuracy in Blocks World scenarios but continuing to underperform in other planning domains. These findings raised important questions about whether these improvements were due to improved approximate retrieval or genuine planning abilities.

The critical question in this research is not whether LLMs can simulate reasoning by leveraging their memory and pattern recognition capabilities, but whether they can perform genuine principled reasoning. Although pattern recognition is valuable in many contexts, it fundamentally differs from true reasoning, which requires deriving solutions from first principles rather than recalling similar patterns from training data. This distinction becomes increasingly challenging to identify as models are trained on increasingly vast datasets, creating a significant concern for educators, evaluators, and researchers attempting to assess LLMs' true capabilities.

To test whether GPT-4's improved performance in planning tasks resulted from enhanced approximate retrieval or actual planning abilities, researchers employed a clever methodology: obfuscating the names of actions and objects in planning problems to reduce the effectiveness of pattern matching and retrieval [3]. The results revealed that GPT-4's performance dropped sharply under these conditions, while standard AI planning systems remained unaffected. This experiment proves that GPT-4 relies more heavily on pattern recognition than true planning capabilities.

One promising approach to improving LLM planning is the implementation of an external model-based plan verifier for back-prompting and solution certification [3]. The LLM-Modulo framework exemplifies this approach, combining LLMs' idea generation capabilities with external verifiers in a systematic generate-test-critique process, which helps ensure correctness in reasoning tasks. This represents a more principled approach than relying solely on the LLM's internal capabilities.

In contrast, the more common human-in-the-loop method often falls prey to the Clever Hans effect, where the LLM essentially makes educated guesses, and the human, who already knows the correct solution, guides it towards the right answer through iterative prompting as shown in Figure 2.1. This approach can create a misleading impression of the LLM's intrinsic reasoning abilities, particularly in scenarios where humans cannot independently verify the solution's correctness.

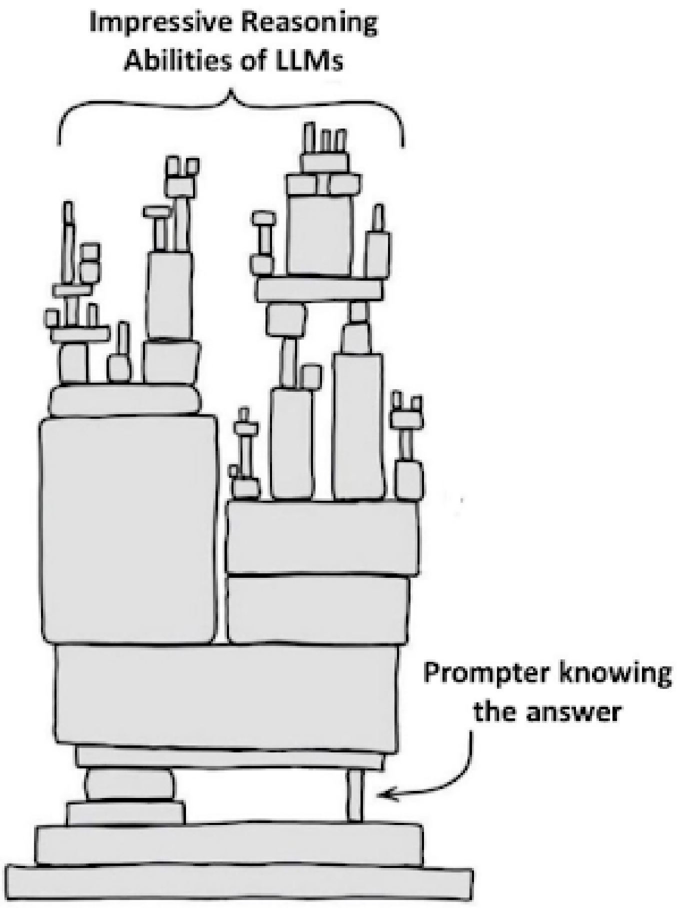

Impressive Reasoning Abilities of LLMs
Prompter knowing the answer

Figure 2.1: Claimed reasoning capabilities of LLMs are often due to the subconscious helpful prompting by the humans in the loop. Image from "Can Large Language Models Reason and Plan?"[7]

2.2 Enhancing LLMs' Reasoning and Planning Capabilities

LLMs may not be capable of autonomous reasoning or planning straight out of the box, but there are techniques to "nudge" them towards better performance in this area.

Two popular methods for enhancing LLM planning abilities are:

- **Fine-Tuning:** This involves training a general LLM on specific planning problems and their solutions to improve its predictions. While fine-tuning may enhance performance with more data, it primarily transforms the task into memory-based approximate retrieval, not true planning. It doesn't prove independent planning abilities.

- **Back-Prompting:** This approach provides hints or suggestions to guide the LLM towards better plans. Key considerations include whether the prompts are manual or automated, who certifies correctness, and whether the prompts inject domain-specific knowledge or encourage further attempts. While back-prompting may lead to better performance, it still depends on the quality and structure of the prompts used.

Neither method, in isolation, demonstrates that LLMs can autonomously plan; instead, they shift the task towards pattern recognition and retrieval. The key takeaway is that LLMs are excellent for idea generation and approximate retrieval, which can support reasoning and planning in a more structured framework, without attributing questionable planning capabilities to the models themselves.

Chapter 3

Challenges and Mitigations

3.1 Challenges in Reasoning with LLMs

Despite the remarkable progress in Large Language Models' capabilities, their ability to perform reliable reasoning remains a significant challenge. A few of these challenges, which were highlighted in recent research[8], are outlined below:

Lack of True Reasoning Abilities: LLMs are good at generating text based on patterns learned during training, but don't truly "reason" in the traditional sense. Their answers are based on statistical associations, not logical deductions, which makes them prone to errors in complex reasoning tasks. *Example:* When asked, "If Alice is older than Bob and Bob is older than Carol, who is the oldest?" an LLM may occasionally give an incorrect answer like "Bob," failing basic transitive logic.

Grounded Analogies: LLMs struggle with analogies that require grounding in real-world or physical knowledge, limiting their ability to apply reasoning in contexts outside the scope of their training data. *Example:* When prompted with "An aeroplane is to sky as a submarine is to," the model might respond with "boat" instead of "ocean".

Cross-Domain Complexity: Bridging semantically distant domains remains challenging for LLMs, as they lack integrated multimodal learning and struggle to transfer knowledge across vastly different domains. *Example:* Combining biological concepts with computer science, such as drawing parallels between DNA replication and version control, may yield incoherent responses.

Explainability: LLMs do not inherently explain their reasoning processes, which hampers their interpretability, particularly in complex analogical tasks where understanding the reasoning is crucial. *Example:* When solving a riddle, an LLM might give the correct answer but offer no valid reasoning path, or invent one.

Staleness: LLMs may provide outdated information, as their training data may not be current, leading to errors when information evolves. *Example:* A model trained before 2023 might respond that "India has no UPI-based international transactions," despite recent developments.

Mathematical Incorrectness: LLMs do not directly perform mathematical operations like a calculator, resulting in inaccurate or inconsistent mathematical outputs in complex calculations. *Example:* The "LLM strawberry test" refers to the challenge where language models struggle to correctly count the number of "r" s in the word "strawberry." Despite their high-level language abilities, these models can miss fine details due to tokenisation and character-level processing limitations.

Inclination to Hallucinate: LLMs are trained to generate plausible-sounding text, even if they do not know the correct answer. This makes them prone to hallucination, where they fabricate information rather than provide factually accurate results. *Example:* When asked

for a source on a niche legal loophole, an LLM might invent a non-existent article or author.
Weaknesses in Multi-Step Problem-Solving: LLMs face significant challenges in multi-step problem-solving due to their difficulty maintaining logical coherence across reasoning steps, particularly in complex tasks like mathematics. Mistakes made in earlier steps often propagate, compounding errors and leading to incorrect results. *Example:* When solving a multi-step algebra problem, a small mistake in isolating a variable in the first step can derail the entire solution.
Context Preservation: While LLMs can track context over short conversations or text inputs, they struggle with preserving context over longer documents or multi-turn dialogues, which is crucial for complex reasoning tasks that involve a larger body of knowledge. *Example:* In multi-turn reasoning tasks where earlier context establishes critical definitions or constraints, models often forget these details in later turns, leading to inconsistent application of rules. Boye et al.[9] observed this in problems requiring application of previously defined terms, with models reinterpreting or ignoring earlier definitions as the reasoning chain progressed.

3.2 Improvements for Reasoning

While Large Language Models exhibit impressive language capabilities, their reasoning often lacks robustness and accuracy. To address these limitations, several techniques have emerged that enhance LLMs' ability to reason more reliably. This section explores three prominent approaches: Retrieval-Augmented Generation, Program-Aided Language Models, and Agentic Chain-of-Thought, each offering distinct mechanisms for improving factual correctness, logical consistency, and multi-step problem solving.

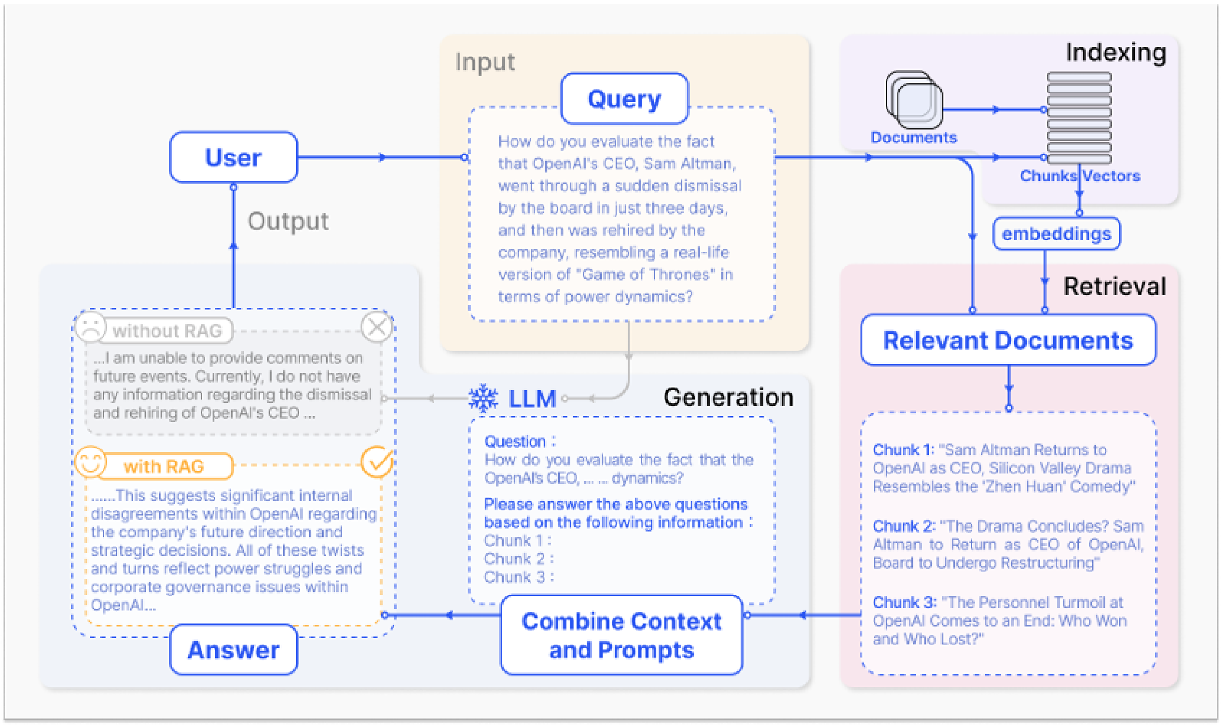
3.2.1 Retrieval-Augmented Generation

Retrieval-Augmented Generation enhances LLM responses by integrating external knowledge sources during inference. Instead of generating answers from internal parameters, RAG retrieves relevant context from a document collection and combines it with the model's response generation. This retrieval step typically uses vector similarity to find passages related to the input query. These passages are then appended to the prompt or processed jointly to produce more accurate and grounded responses. The overall RAG workflow is illustrated in Figure 3.1, which includes indexing documents into a vector store, retrieving the top-k relevant chunks, and passing both the query and the retrieved information to the LLM. RAG is handy for improving factual correctness and enabling domain-specific applications, such as customer support, biomedical QA, and technical documentation querying. However, basic RAG still faces challenges like retrieving irrelevant or redundant documents, and models may sometimes over-rely on noisy contexts.

An extension of basic RAG, GraphRAG enhances retrieval by representing information as structured knowledge graphs rather than raw text. Nodes represent entities or concepts, and edges capture their relationships[11]. When answering queries, relevant subgraphs are retrieved instead of flat documents, enabling more structured and interpretable reasoning. GraphRAG is particularly useful in domains where relationships and hierarchies are essential, such as scientific research or legal document analysis.

3.2.2 Program-Aided Language Models

Program-Aided Language Models leverage the unique strengths of large language models for problem decomposition and deterministic code execution[12]. Unlike traditional LLMs that



How do you evaluate the fact that OpenAI's CEO, Sam Altman, went through a sudden dismissal by the board in just three days, and then was rehired by the company, resembling a real-life version of "Game of Thrones" in terms of power dynamics?

...I am unable to provide comments on future events. Currently, I do not have any information regarding the dismissal and rehiring of OpenAI's CEO ...

......This suggests significant internal disagreements within OpenAI regarding the company's future direction and strategic decisions. All of these twists and turns reflect power struggles and corporate governance issues within OpenAI...

Question :
How do you evaluate the fact that the OpenAI's CEO, ... dynamics?
Please answer the above questions based on the following information :
Chunk 1 :
Chunk 2 :
Chunk 3 :

Chunk 1: "Sam Altman Returns to OpenAI as CEO, Silicon Valley Drama Resembles the 'Zhen Huan' Comedy"

Chunk 2: "The Drama Concludes? Sam Altman to Return as CEO of OpenAI, Board to Undergo Restructuring"

Chunk 3: "The Personnel Turmoil at OpenAI Comes to an End: Who Won and Who Lost?"

Figure 3.1: A representative instance of the RAG process applied to question answering, adapted from "Retrieval-Augmented Generation for Large Language Models: A Survey" [10]. It mainly consists of 3 steps. 1) Indexing: Documents are split into chunks, encoded into vectors, and stored in a vector database. 2) Retrieval: Retrieve the Top-k chunks most relevant to the question based on semantic similarity. 3) Generation: Input the original question and the retrieved chunks together into the LLM to generate the final answer.

rely on probabilistic inference and may make arithmetic or logic errors, PALs aim to bridge this gap by coupling the language model's ability to understand complex tasks with an executable code generation capability that guarantees correctness. PAL models excel at breaking down complex problems into smaller, manageable subproblems, making them particularly useful for tasks involving sequential reasoning or multi-step calculations. When generating code (such as Python scripts), PAL models ensure that the logic is executable, avoiding common pitfalls like miscalculations, incorrect syntax, or other logical errors that can occur in traditional language model-based solutions.

Frameworks like *THINK-AND-EXECUTE* further enhance PAL's effectiveness by introducing intermediate layers where pseudocode or algorithmic abstractions are generated before final code is written[13]. This step-by-step breakdown allows the model to refine its approach and align more closely with the logic needed for solving the problem. Additionally, by generating pseudocode or high-level descriptions before diving into actual code, PAL models can achieve better performance even when smaller in size compared to traditional LLMs, making them particularly attractive for resource-constrained environments. Hybrid techniques such as *PoT* use language models to generate text, programming language statements, and finally an answer. In PoT, the computation can be delegated to a program interpreter, which is used to execute the generated program, thus decoupling complex computation from reasoning and language understanding[14].

PAL's strengths are particularly evident in fields requiring both high-level reasoning and accuracy, such as constraint satisfaction problems (CSP), optimisation tasks, and technical domains like numerical analysis, machine learning, and algorithm design. In summary, PAL offers a new paradigm in which the power of LLMs is complemented by deterministic execution of code, enhancing performance in critical domains where accuracy and reliability are paramount.

3.2.3 Agentic Chain-of-Thought

Chain-of-Thought prompting is a simple yet powerful technique that enhances reasoning in large language models by guiding them to generate intermediate steps towards a final answer. Instead of jumping directly to a conclusion, the model is shown a few cues where reasoning is made step by step. This approach significantly improves performance on complex tasks such as mathematics word problems, logic puzzles, and multi-hop questions. Remarkably, such reasoning abilities tend to emerge naturally in sufficiently large models when prompted appropriately, without requiring fine-tuning or architectural changes.

Extending standard Chain-of-Thought prompting, agentic implementations introduce autonomous agents that can plan, reason, and act over multiple steps using external tools, APIs, or memory stores. Unlike static prompting, these agents engage in iterative refinement through feedback loops, tool usage, and self-verification, enabling more structured and context-aware problem solving.

In typical agentic systems, each agent is assigned a specific role (e.g., planner, solver, verifier) and is responsible for subtasks within a larger reasoning pipeline. The system orchestrates multiple agents to decompose complex queries into manageable components, enabling parallel or sequential execution. Tasks such as multi-step mathematical problem solving, document-based question answering, and code generation benefit from this decomposition, as intermediate outputs can be verified or refined before finalising an answer.

Agentic CoT with access to external resources (such as search engines or APIs) has demonstrated significantly improved performance in detecting hallucinations and verifying factual consistency. However, this robustness comes with computational trade-offs, typically resulting in significant overhead due to repeated reasoning steps and sequential API/tool calls.

3.2.3.1 Agent Frameworks

Several frameworks simplify the construction and orchestration of agent-based systems. One notable example is **CrewAI**[15], an open-source agent orchestration framework that enables developers to create multi-agent pipelines with well-defined roles and tasks. CrewAI supports modular development, allowing agents to collaboratively communicate, share memory, and coordinate actions to solve complex problems. These systems bring structured reasoning workflows closer to general-purpose AI agents by allowing tools like code interpreters, retrievers, or web browsers to integrate into the reasoning loop seamlessly.

3.2.3.2 Concepts associated with Agentic Frameworks

Understanding agentic systems requires familiarity with a few foundational concepts:

- **Task:** A specific objective or problem to be solved, such as solving a mathematics question or summarising a document.
- **Tool:** An external capability like a code interpreter, web browser, or database retriever that agents can invoke to assist with reasoning.

- **Agent:** An autonomous component assigned a role (e.g., planner, solver, verifier) that performs subtasks using reasoning and tools.
- **Process:** The overall reasoning workflow, often involving planning, decomposing tasks, invoking tools, and verifying outputs iteratively.
- **Crew:** A coordinated group of agents working collaboratively to complete complex reasoning tasks, typically orchestrated through a framework.
- **CrewAI:** An open-source framework for building and orchestrating multi-agent systems, enabling structured reasoning via modular design.

3.2.3.3 Emerging Paradigms

Techniques such as **Meta Agent Search**[16] push agentic reasoning further by automatically evolving the structure of reasoning workflows through reinforcement or evolutionary methods. These systems discover optimal reasoning patterns that outperform human-engineered prompts in domains like mathematics, programming, and logic-based question answering.

Chapter 4

Methodology

This chapter delves into the comprehensive experimental framework employed to evaluate and compare the reasoning capabilities of Large Language Models. The methodology encompasses a diverse set of reasoning tasks, evaluation techniques, external knowledge bases, and reasoning enhancement techniques designed systematically to analyse the performance of different models while handling complex logical reasoning problems.

We start by establishing our evaluation approach's foundation, detailing the domains of test problems used to query the models. We also provide illustrative sample cases that demonstrate the reasoning complexity associated with each domain. We then describe the knowledge corpus constructed to support different reasoning methodologies, like the question bank for Retrieval-Augmented Generation and the pseudocode collection for Program-Aided Language Models. These carefully curated knowledge bases serve as the external information sources that models can leverage when applying various reasoning enhancement techniques.

The core of our analysis involves ten distinct reasoning approaches across three major categories. First, we examine Direct Response Generation techniques, including baseline responses without additional support and RAG implementations with and without explanatory content. Second, we explore Program-Aided Language Models using both Python and Resource Description Framework (RDF) code generation, evaluating implementations from scratch, static one-shot prompting, and dynamic few-shot approaches. Finally, we assess Agentic Chain-of-Thought methodologies, comparing performance with and without internet access to understand how external knowledge retrieval affects reasoning quality.

We implement specialised evaluation techniques to assess performance across these methodologies, including a Regex Extractor for parsing model outputs and a PAL Executor for validating generated code. These tools enable consistent scoring and comparison across models and approaches, ensuring fair assessment of reasoning capabilities.

Our experimental design facilitates three levels of comparative analysis: methodology benchmarking (comparing different reasoning enhancement techniques), architectural comparisons (examining open-source versus closed-source models, parameter scaling effects, and the impact of specialised training), and generalisation assessment (evaluating performance consistency across multiple runs). This multi-dimensional evaluation framework provides a comprehensive view of current LLM reasoning capabilities and the effectiveness of various approaches for enhancing logical inference in these systems.

4.1 Reasoning Questions

In this section, we outline the test problems and the domains associated with these problems. We also display the diversity of the problem pool by conducting a clustering analysis on the dataset. Sample questions related to each domain are also provided to demonstrate the difficulty associated with each reasoning problem.

4.1.1 Domains of Test Problems and Representative Samples

To comprehensively evaluate LLM reasoning capabilities, we curated a diverse set of 100 complex reasoning problems spanning multiple domains and difficulty levels. We employed vector clustering to validate the diversity of our question set, ensuring broad coverage of reasoning types. Our question set includes problems drawn from established benchmarks used in state-of-the-art reasoning model evaluations, such as American Invitational Mathematics Examination (AIME)[17] and MATH-500[18], which have become standard metrics for assessing advanced reasoning capabilities in recent models. These benchmarks are particularly valuable as they represent challenging problems requiring multi-step deduction and creative problem-solving approaches, similar to those used to evaluate DeepSeek-R1[19]. The reasoning domains encompassed by our question set are given below.

4.1.1.1 CAT Logical Reasoning and Data Interpretation

These questions are extracted from the previous year papers of the Common Admission Test (CAT)[20], a computer-based test for admission to graduate management programs in India. Logical Reasoning and Data Interpretation is one of the sections in the paper which tests deductive and analytical reasoning through scenarios requiring careful constraint satisfaction. A representative example involves the following seating arrangement problem:

Exactly six trade representatives negotiate a treaty: Klosnik, Londi, Manley, Neri, Osata, Poirier. There are exactly six chairs evenly spaced around a circular table. The chairs are numbered 1 through 6, with successively numbered chairs next to each other and chair number 1 next to chair number 6. Each chair is occupied by exactly one of the representatives. The following conditions apply: Poirier sits immediately next to Neri. Londi sits immediately next to Manley, Neri, or both. Klosnik does not sit immediately next to Manley. If Os to Manley. If Londi sits immediately next to Poirier, which one of the following is a pair of representatives who must sit immediately next to each other?

(A) Klosnik and Osata

(B) Londi and Neri

(C) Londi and Osata

(D) Manley and Neri

(E) Manley and Poirier

4.1.1.2 AIME Benchmark Problems (1986–2005)

The American Invitational Mathematics Examination (AIME) is a prestigious competition containing challenging problems requiring advanced mathematical reasoning. These problems are particularly valuable for evaluating LLMs' multi-step mathematical reasoning capabilities. A sample problem from this benchmark:

A biologist wants to calculate the number of fish in a lake. On May 1, she catches a random sample of 60 fish, tags them, and releases them. On September 1, she catches a random sample of 70 fish and finds that 3 of them are tagged.

> *To calculate the number of fish in the lake on May 1, she assumes that 25% of these fish are no longer in the lake on September 1 (because of death and emigrations), that 40% of the fish were not in the lake on May 1 (because of births and immigrations), and that the number of untagged fish and tagged fish in the September 1 sample are representative of the total population. What does the biologist calculate for the number of fish in the lake on May 1?*
>
> *(A) 800*
> *(B) 810*
> *(C) 820*
> *(D) 840*
> *(E) 880*

4.1.1.3 MATH500 Benchmark

MATH500 is a benchmark specifically designed for evaluating LLM mathematical reasoning capabilities. This dataset contains a subset of 500 problems from the MATH benchmark that OpenAI created in their publication "Let's Verify Step by Step" [21]. It includes problems across various mathematical domains, including algebra, calculus, geometry, statistics, and number theory, with carefully calibrated difficulty levels. A representative example:

> *A palindrome is a number that reads the same forwards and backwards. The sum of a particular set of three consecutive positive integers is a three-digit palindrome. If the sum is less than 220, what is the greatest possible value for the largest of the three integers in the set?*
>
> *(A) 50*
> *(B) 55*
> *(C) 56*
> *(D) 58*
> *(E) 60*

4.1.1.4 Probability Problems

These questions test statistical reasoning and the ability to compute probabilities across different random processes. This section involves various domains like coin toss, deck of cards, dice problems, etc. An example is given below :

> *If two six-sided dice are rolled, what is the probability that the product of the numbers rolled is greater than 15?*
>
> *(A) 1/4*
> *(B) 5/18*
> *(C) 11/36*
> *(D) 1/3*
> *(E) 13/36*

The diversity of our question set ensures a comprehensive evaluation of LLM reasoning abilities across multiple domains, allowing us to identify both general reasoning capabilities and domain-specific strengths or weaknesses.

4.1.2 Vector Clustering Analysis

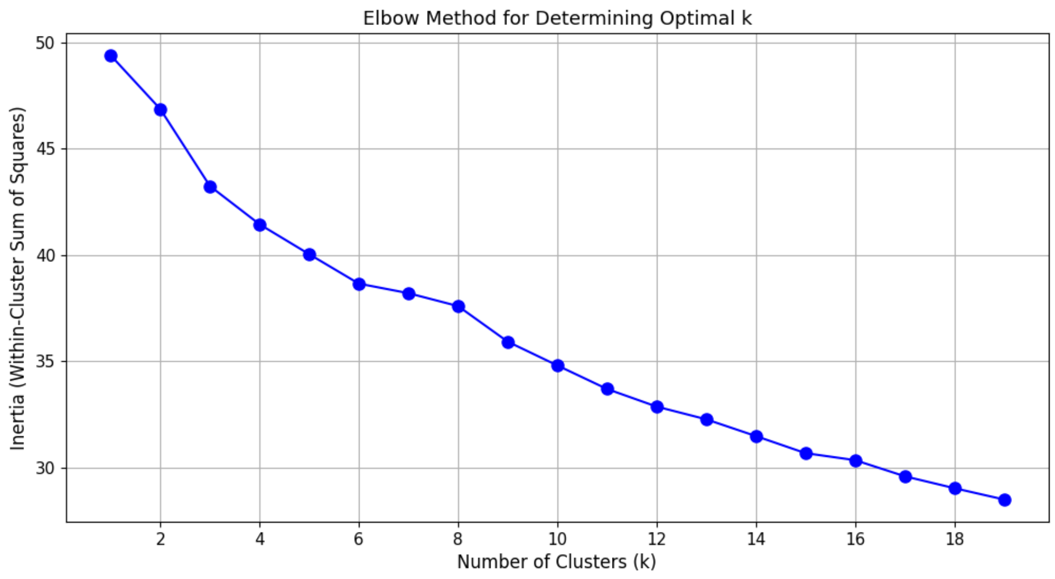


Figure 4.1: Elbow method for determining optimal number of clusters.

We performed embedding-based clustering analysis to assess the diversity of our question dataset. First, we generated vector embeddings for all 100 questions using the Ollama model `nomic-embed-text:latest`, resulting in a high-dimensional representation of size (100, 768). To visualise these embeddings, we applied Principal Component Analysis (PCA) to reduce the dimensionality to the first two principal components.

We then conducted k-means clustering on these reduced vectors to identify natural groupings of similar question types. To determine the optimal number of clusters, we created an elbow plot (Figure 4.1) that plots the within-cluster sum of squares against different values of k. The point where the curve forms an "elbow" indicates the optimal cluster count. Such a deviation is observed at 6 clusters.

The resulting k-means visualisation (Figure 4.2) revealed distinct clusters corresponding to different reasoning domains, with centroids (marked with **X**) representing the typical question for each category. This clustering confirmed that our question set adequately spans diverse reasoning tasks rather than concentrating on any single domain.

4.2 Knowledge Corpus

This section details the curated knowledge corpora developed to support our reasoning evaluation methodologies. These corpora serve as a primary information source that enables different reasoning enhancement techniques, providing templates that models can utilise when tackling complex problems. Distinct corpus types optimised for different reasoning approaches were created, each designed to explore specific aspects of how external knowledge influences LLM reasoning capabilities. The following subsections describe these specialised collections, their structure, and their intended application within our experimental framework.

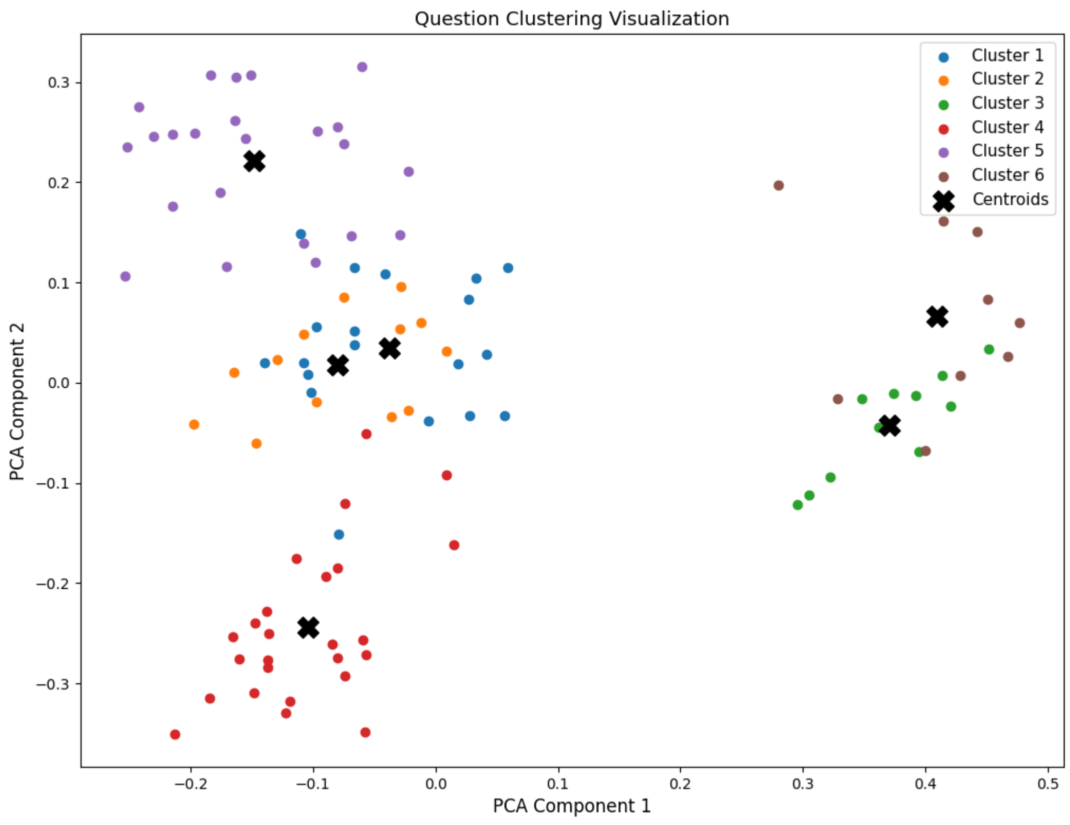


Figure 4.2: K-means clustering of reasoning questions with centroids.

4.2.1 Knowledge Corpus to support RAG

The knowledge corpus for Retrieval-Augmented Generation consists of two primary document collections, each serving different purposes in our experimental framework.

4.2.1.1 RAG without Explanations for Solutions

The first corpus comprises a collection of 60 reasoning problems presented without logical explanations for the solution. This collection includes only problem statements, answer choices, and correct answers without intermediate steps or reasoning chains.

For instance, problems like *"When a six-sided die is rolled, what is the probability of getting a prime number?"* are presented with only the final answer (1/2) rather than the step-by-step calculation. This approach allows for more examples within the same context window, as each example requires less space.

This corpus serves as a comparative condition in our experiments, enabling us to assess the impact of simple answer retrieval versus detailed explanation retrieval on model performance. It helps evaluate whether models benefit more from exposure to a larger quantity of examples or from fewer examples with comprehensive reasoning steps.

4.2.1.2 RAG with Explanation for Solutions

The second corpus contains 60 complex reasoning problems with detailed step-by-step explanations. These problems span multiple domains, including logical puzzles (circular seating arrangements, scheduling constraints), probability questions (dice rolls, card draws), and constraint satisfaction problems. Each problem includes the correct answer and comprehensive explanations that walk through the entire reasoning process.

The explanations detail how to systematically apply each constraint, eliminate invalid arrangements, and derive the correct solution. This rich contextual information enables the RAG system to retrieve similar problems and detailed reasoning patterns that can be adapted to new questions.

4.2.2 Knowledge Corpus to support PAL

Our PAL framework relies on two specialised corpora that provide different programming paradigms for implementing automated reasoning.

4.2.2.1 Python-Based PAL Corpus

The Python-based PAL corpus contains reusable code templates and function implementations for solving various reasoning problems programmatically. This collection includes functions for calculating probabilities, handling permutations and combinations, and processing constraint satisfaction problems.

The code is structured for modularity and reuse, with example implementations for common problem types such as dice roll outcomes, card draws, and logical constraints. For example, the corpus includes generic functions that can be applied across multiple card-drawing scenarios. This programmatic approach enables LLMs to generate executable Python code that leverages computational precision for complex probabilistic reasoning.

4.2.2.2 RDF-Based PAL Corpus

The RDF-based PAL corpus demonstrates implementations of reasoning problems using the Resource Description Framework (RDF) semantic modelling approach. This collection contains examples of representing logical relationships as RDF triples, defining constraints using SPARQL queries, and performing inference through graph patterns.

These examples showcase how semantic reasoning can be structured and automated through RDF's triple-based knowledge representation, offering a different programming paradigm than procedural Python code for implementing logical inference in LLMs.

4.3 Methods to Aid Reasoning

4.3.1 Direct Response Generation

This section explains the process behind large language models attempting to solve reasoning tasks independently without the support of external tools like code executors or agentic frameworks. In this methodology, we provide the models with a problem, and the model is prompted to produce a step-by-step solution and select an answer from the option pool. This method acts as the primary baseline to compare the performance of other complex methodologies. Besides the baseline method, we utilise Retrieval-Augmented Generation to enhance LLM performance by providing external knowledge chunks during inference. Models may access relevant examples from the external corpus to guide their reasoning.

4.3.1.1 Baseline Response

The baseline response method involves directly querying a large language model without augmentation, explanation, or prompt engineering. While this approach is fast and straightforward, its reasoning capabilities are limited. The model generates responses based solely on patterns learned during training and lacks any explicit logical reasoning. This may lead to superficial answers and hallucinations, especially for tasks requiring multi-step reasoning, real-world grounding, or precise logic. But we may use this as the baseline to compare and evaluate the performance of other methodologies like RAG, PAL, etc.

4.3.1.2 Example Solutions Without Explanation for RAG

In this approach, Retrieval-Augmented Generation obtains relevant answers from an external knowledge base, without incorporating intermediate reasoning or explanations. The model is guided towards correct answers through retrieved examples, but lacks insight into the reasoning behind them. As a result, this setup offers better factual grounding than the baseline but still struggles with tasks that require understanding or generalisation beyond surface-level patterns. This methodology can set up a grounding for the LLM to compare and verify whether its reasoning is correct by verifying the solution to the problems extracted from RAG. We will have a dataset of 60 questions available, and their solutions will be provided as context in this study.

4.3.1.3 Example Solutions With Explanation for RAG

This variant of RAG enhances the reasoning capabilities of the model by providing examples that include detailed explanations or step-by-step solutions. By exposing the model to how reasoning unfolds, it learns not just the correct answers but the logic used to arrive at them. This dramatically improves generalisation to novel problems and supports more reliable and interpretable reasoning. It is particularly effective for tasks like analogical reasoning and multi-hop question answering. In this setup, we use the above-mentioned dataset of 60 questions, but instead of just the solution, here we also provide the logical reasoning behind the problems involved.

4.3.2 Program-Aided Language Models

In this section, we delve into the methodologies which incorporate code generation and symbolic program execution instead of natural language generation to enhance the reasoning abilities of LLMs. The PAL methodology is based on the idea of generating a program which, on execution, produces a final answer. In this framework, since models produce programmatic code, the burden on the model is reduced to simulate logical reasoning steps. We explore the effects of static one-shot as well as dynamic few-shot prompting. The impact of using a Python Reasoning Engine like Resource Description Framework to capture structured knowledge graphs for logic-based interpretation is also explored.

4.3.2.1 PAL from Scratch for Python Code

Program-Aided Language Models from scratch involve prompting the LLM to generate executable Python code without in-context examples. This method improves reasoning significantly by externalising complex calculations or logic to a Python interpreter. Even without examples, if the model has been sufficiently trained on code, it can produce syntactically correct and logically sound outputs. It excels in domains like mathematics, data manipulation, and algorithmic reasoning. This method is enabled by engineering the prompt to provide readily executable Python code from the LLM.

4.3.2.2 PAL from Scratch for RDF Code

This setup uses PAL to produce symbolic or structured outputs in Resource Description Framework format instead of Python. While it follows a similar reasoning paradigm where symbolic logic is offloaded to a deterministic interpreter, it is more challenging for models due to RDF's lower representation in training data. As a result, reasoning here is possible but less robust, and often requires additional training or alignment to achieve meaningful output.

4.3.2.3 PAL with Static One-Shot Prompting for RDF Code

Here, the PAL model is guided by static, predefined examples of RDF code. These examples act as templates for reasoning, helping the model to mimic structured logic flows when solving similar problems. While this improves reasoning over the scratch variant, it is limited by the fixed nature of the examples. The model may fail to adapt well to queries that deviate significantly from the given examples, but it performs reliably within known patterns. It can derive insights from the provided sample code to understand the syntax involved with RDF coding.

4.3.2.4 PAL with Dynamic Few-Shot Prompting for Python Codes

This technique significantly enhances reasoning by dynamically retrieving relevant Python code examples based on the current query. The LLM uses these contextual cues to generate and execute new code that logically solves the problem. This method combines the strengths of retrieval, reasoning, and symbolic execution, and is highly effective for complex numerical, logical, or rule-based tasks. It supports generalisation while maintaining consistency and correctness in reasoning steps.

4.3.2.5 PAL with Dynamic Few-Shot Prompting for RDF Codes

Like the previous method, using RDF code instead of Python, this approach dynamically fetches reasoning examples that help the model produce structured symbolic logic. It enhances the model's ability to operate in knowledge graphs or semantic reasoning settings. Though not as natural as Python for most LLMs, it allows for precise logical relationships and can support ontology-driven reasoning when paired with robust RDF examples.

4.3.3 Agentic Chain-of-Thought

Agentic Chain-of-Thought methodologies represent advanced reasoning frameworks that enable LLMs to decompose complex problems into explicit step-by-step reasoning paths while maintaining consistency through the solution process. This section examines two variants of this approach: a closed-system implementation that relies solely on internal knowledge and computational tools, and an internet-enabled implementation that can leverage external information sources and verification mechanisms.

4.3.3.1 Agentic CoT Methodologies Without Internet

In this setting, the model follows a step-by-step, agent-like reasoning path, but operates within a closed environment without external data access. It uses internal tools like calculators and memory modules to solve problems. This approach balances coherence and control, supporting structured reasoning across multiple steps. While it lacks real-time information access, it is well-suited for benchmark tasks involving logical deduction, math, or procedural understanding.

4.3.3.2 Agentic CoT Methodologies With Internet

With internet access, Agentic Chain-of-Thought empowers the model to reason step-by-step while interacting with external tools like search engines, calculators, or APIs. This agent-like behaviour allows the model to fetch current information, validate facts, and chain intermediate steps effectively. It represents one of the most advanced forms of reasoning, capable of handling open-ended, real-time tasks requiring logic and knowledge discovery.

4.4 Evaluation Techniques

To systematically assess LLM performance across diverse reasoning tasks, we developed specialised evaluation tools that automate the extraction and analysis of model outputs. These techniques ensure consistent scoring methodology across different models and reasoning approaches, enabling fair comparison of reasoning capabilities. This section details two complementary evaluation mechanisms: the Regex Extractor, which identifies and standardises final answers from natural language responses, and the PAL Executor, which safely extracts and runs code snippets generated by models to verify computational correctness. Together, these automated evaluation techniques facilitate reproducible assessment of direct answer generation and code-based reasoning approaches.

4.4.1 Regex Extractor

The Regex Extractor is a utility function that processes and extracts relevant answers from textual responses generated by language models. It uses regular expressions to search for specific patterns in the text that typically indicate the final answer, such as "Final response is: ", "The answer is: ", or "The correct answer is: ". Additionally, it can detect LaTeX boxed answers, and if no predefined pattern is found, it attempts to extract the last capitalised word or number from the response. The extracted answer is cleaned by removing unnecessary characters like quotes or special symbols. To ensure accurate comparison with the correct answer, number words (e.g., converting "zero" to "0") and other variations are accounted for. The extracted answers and their matching scores are saved in JSON and CSV formats, with special handling to ensure that values are correctly formatted for compatibility with tools like Excel, avoiding misinterpretation of fractions and dates.

4.4.2 PAL Executor

The PAL Executor is a Python-based system designed to extract and execute Python code embedded in textual responses generated by language models. It first identifies Python code blocks within the response, either in markdown format or by detecting code-like structures such as 'import' or 'def' statements. Once the Python code is extracted, the system executes it in a controlled environment with predefined global variables, ensuring the execution is safe and limited. The function runs the extracted code, and if it completes successfully, the result is returned; otherwise, errors are captured, and timeouts are handled. After execution, the predicted answer is compared with a reference answer using another function, which allows for flexible matching, including handling fractional numbers, lists, and approximate matches.

Finally, the results, including the extracted code, predicted answer, and comparison score, are stored in JSON and CSV formats for easy review and analysis.

4.5 Studies

This section presents the experimental studies of reasoning evaluations to be conducted across various large language models (LLMs) and prompting methodologies. Each subsection analyses different aspects of the study, from model performance to the repeatability of results.

4.5.1 Methodology Benchmarking

4.5.1.1 RAG with explanation vs RAG without explanation

We compared the performance of RAG methodologies with and without explanations. Here, we intend to understand the middle-ground between the number of questions provided and the rigour of logical reasoning steps provided. For a similar chunk size, the explanation-based setup provides 1-2 questions, while without explanations, we may accommodate more than 10 questions at a time. This study will investigate whether more reasoning cues enhance reasoning quality than a larger basket of questions without explanation.

4.5.1.2 Static One-Shot vs Dynamic Few-shot Adaptation

We compared static prompting with dynamic few-shot adaptation to assess the adaptability of each method in handling reasoning tasks. Static prompting may work well for simpler tasks, but could be less effective in complex scenarios. Dynamic adaptation allows the model to adjust its reasoning process in real-time based on the context provided.

4.5.1.3 PAL for Python Code Generation vs PAL for Reasoning Engine Frameworks

We compared using PAL for code generation in two frameworks: plain Python and RDF-based Reasoning Engines. RDF Reasoning Engines are well-suited for structured logical reasoning involving relationships between entities. However, LLMs might struggle to generate valid RDF code. This difficulty stems from their limited familiarity with RDF's syntactic conventions and semantic structure. Nonetheless, we explored RDF engines because of their effectiveness in representing and reasoning over structured, relational data, something traditional Python code does not natively support. This analysis aims to determine whether PAL's reasoning strengths can translate across symbolic and procedural domains.

4.5.1.4 Agentic CoT Prompting with and without Internet Access

We compared Agentic Chain-of-Thought prompting with and without internet access. With internet access, CoT models can integrate real-time fact-checking and retrieve additional knowledge from external sources, leading to reduced hallucinations. Without internet access, the models would rely more on their internal knowledge. However, they may lack newer context, which could be helpful in problem-solving. We aim to quantify the reliability and factual grounding benefits that internet-enabled prompting may provide.

4.5.2 Architectural Comparisons

4.5.2.1 Open-source vs Closed-source Models

We evaluated open-source models (e.g., Llama-3.1, OpenHermes) against closed-source systems (Gemini) to assess their performance on various reasoning tasks. This comparison aims to shed light on different dimensions, such as logical consistency and robustness to prompt variations, potentially providing insights into how model transparency and training scale affect reasoning quality. This study will help inform whether openness in architecture correlates with interpretability or performance trade-offs.

4.5.2.2 Parameter Scaling Effects

The effect of parameter scaling was also tested using model variants with 8B and 70B parameters. Smaller models might struggle with maintaining logical coherence on more complex tasks as they often produce incomplete or inaccurate outputs. Larger models could leverage richer internal representations and broader training contexts to give clear, well-defined answers. This experiment examines whether scaling alone significantly enhances reasoning capabilities across diverse prompting techniques.

4.5.2.3 Specialised Training Effects

We compared models with different training specialisations (conversational, code, and reasoning) to assess how specific training objectives impact reasoning capabilities. These models will be chosen with similar parameter counts to control for size effects. General language models often demonstrate broad capabilities but struggle with structured reasoning or coding tasks. We aim to understand whether domain-specific fine-tuning leads to measurable advantages in structured reasoning tasks.

4.5.3 Generalisation Assessment

4.5.3.1 Performance Drift Across Multiple Runs

We assessed the repeatability of reasoning tasks and performance drift across multiple experimental runs. The capability of different models to produce similar solutions over multiple runs was evaluated for different models in the baseline setup.

To study the generalisation and consistency of LLMs in reasoning tasks, we conducted multiple evaluation runs using a fixed set of logical reasoning questions under the same baseline setup. By comparing model predictions across two independent runs, we assessed both the stability of their outputs and the likelihood of performance drift. This analysis helps reveal how reliably a model maintains its reasoning capabilities over repeated executions, a key feature in applications demanding consistent reasoning.

Chapter 5

Direct Response Generation

Direct response generation refers to the process where large language models attempt to solve reasoning tasks independently, without the assistance of external tools, specialised prompting strategies such as Program-Aided Language Models or Chain-of-Thought prompting. This setup provides models with a problem and its associated options. The model is prompted to produce the complete step-by-step solution in a single attempt. Only internal knowledge and reasoning capabilities are utilised to reach a conclusion. This method provides a fundamental baseline to assess the native reasoning strength of LLMs without any augmentation. It helps highlight the inherent differences between models of varying architectures, scales, and training data.

Retrieval-Augmented Generation is an approach that enhances LLM performance by providing external knowledge chunks during inference. In this methodology, models access relevant examples or knowledge to guide their reasoning rather than relying solely on their training knowledge. RAG helps mitigate hallucinations and knowledge gaps by grounding responses in retrieved information. This chapter examines two variants of RAG: one in which models are provided with example solutions without explanations and another in which examples include step-by-step explanations.

5.1 Baseline Response

This section presents the baseline performance of different large language models on reasoning tasks using the direct response generation setting. The models evaluated are Gemini Flash 1.5, Llama-3.1 8B, Llama-3 70B, DeepSeek-LLM 7B, DeepSeek-coder 6.7B and DeepSeek-R1 7B. Each model was tasked with solving reasoning problems directly, without any example guidance, fine-tuning, or CoT prompting. Their performances were scored out of 100. In cases where multiple runs were conducted, the lower score was selected to represent each model's minimum capability, providing a more conservative performance estimate. The final scores are summarised in Table 5.1.

Several observations can be drawn from these results. Closed-source models like Gemini Flash 1.5 demonstrated stronger baseline reasoning abilities than open-source models. Despite the parameter size difference between Llama-3.1 8B and Llama-3 70B, there was only minimal improvement in scores, suggesting that scaling beyond a certain point may yield minimal returns for baseline reasoning tasks. DeepSeek-LLM 7B struggled significantly, reaching the lowest score. The low score can be attributed to the inability of DeepSeek-LLM to reach coherent conclusions at the end. It performs very poorly compared to the recently famous DeepSeek-R1. However, the inference time of DeepSeek-R1 is very high, taking around 25 minutes to solve each problem. Due to this, we are only comparing the capabilities of

Model	Score (/100)
Gemini Flash 1.5	78
Llama-3.1 8B	52
Llama-3 70B	55
DeepSeek-LLM 7B	14
DeepSeek-coder 6.7B	11
DeepSeek-R1 7B	61

Table 5.1: Baseline scores for different LLMs.

DeepSeek-LLM, DeepSeek-coder and DeepSeek-R1 for the baseline technique. This helps us compare the importance of task-intensive training for LLM performance.

Several important conclusions can be drawn about the DeepSeek family of models. The dramatic performance gap between DeepSeek-R1 (61) and its siblings DeepSeek-LLM (14) and DeepSeek-coder (11) is particularly noteworthy, especially considering their similar parameter counts. Also, another fact to be noted is that the DeepSeek-coder directly produces code instead of text without explicitly prompting it to do so. These observations about the DeepSeek family reinforce the conclusion that while model scale matters, specialised training targeted at reasoning capabilities can potentially overcome parameter count limitations, though potentially at the cost of inference speed. This makes DeepSeek-R1 an interesting case study of how specialised training can dramatically improve reasoning performance even when model size remains constant.

The results suggest that baseline direct response capabilities depend highly on model scale, training diversity, and optimisation strategies. Furthermore, the large gap between baseline performances indicates the necessity of advanced prompting for achieving competitive reasoning accuracy, especially for smaller or open-source models.

5.2 Example Solutions Without Explanation for RAG

In this section, we evaluate the performance of different large language models when provided with example solutions to similar problems but without detailed explanations of the reasoning process. This approach tests the models' ability to recognise patterns and apply them to new problems without explicit guidance on the underlying logic. The models evaluated include Gemini Flash 1.5, Llama-3.1 8B, Llama-3 70B, and DeepSeek-LLM 7B. Their performances were scored out of 100 and are summarised in Table 5.2.

Model	Score (/100)
Gemini Flash 1.5	85
Llama-3.1 8B	22
Llama-3 70B	54
DeepSeek-LLM 7B	12

Table 5.2: RAG Without Explanation scores for different LLMs.

The findings show notable variations in how models leverage example solutions without explanations. Gemini Flash 1.5 remarkably improved its baseline performance, increasing from 78 to 85, suggesting strong pattern recognition and application capabilities. Conversely, Llama-3.1 8B experienced a significant decline compared to its baseline, decreasing from 52 to 22, indicating potential confusion or over-reliance on explicit guidance. Llama-3 70B maintained relatively consistent performance, slightly decreasing from 55 to 54. DeepSeek-LLM 7B slightly dropped from 14 to 12, though it still performed significantly below other models. Once more, it could still not formulate coherent responses despite adding an external knowledge base.

These findings suggest that the ability to utilise example solutions without explanations effectively varies considerably across models and may depend on model size, training approach, and inherent pattern recognition capabilities. The performance disparity between closed-source and open-source models widens in this setting, highlighting potential differences in their ability to leverage implicit patterns from examples.

5.3 Example Solutions With Explanation for RAG

This section examines how different large language models perform when provided with example solutions that include detailed step-by-step explanations of the reasoning process. This approach tests the models' ability to recognise relevant patterns and understand and apply the underlying logical steps to new problems. We evaluated Gemini Flash 1.5, Llama-3.1 8B, Llama-3 70B, and DeepSeek-LLM 7B. Their performances were scored out of 100 and are summarised in Table 5.3.

Model	**Score (/100)**
Gemini Flash 1.5	77
Llama-3.1 8B	51
Llama-3 70B	58
DeepSeek-LLM 7B	14

Table 5.3: RAG with explanation scores for different LLMs.

The results revealed intriguing patterns in how models utilised examples with logical reasoning steps. Gemini Flash 1.5 showed a decrease in performance compared to the unexplained example approach, dropping from 85 to 77. This suggests that additional explanation may sometimes introduce complexities that interfere with the model's pattern recognition capabilities. In contrast, Llama-3.1 8B showed significant improvement when provided with explanations, increasing from 22 to 51, though still below its baseline performance. This indicates that explicit reasoning paths substantially benefit this model. Llama-3 70B demonstrated a slight performance increase from 54 to 58, suggesting modest benefits from explicit explanations. DeepSeek-LLM 7B showed continued improvement from 12 to 14 but remained the lowest performer.

These findings suggest that the impact of providing explanations with examples varies across models. Smaller or less optimised models like Llama-3.1 8B appear to benefit more from explicit reasoning paths. In contrast, more advanced models like Gemini Flash 1.5 may sometimes perform better with simpler, pattern-based examples. The observed difference may stem from the trade-off between the number of questions and the depth of logical reasoning provided. For the same chunk size, the explanation-based setup provided a maximum of 1-2 questions, while without explanation, we could accommodate around 10-12 questions. This implies that closed-source models might benefit more from exposure to a larger corpus of questions. In contrast, open-source models benefit more from fewer but thoroughly explained examples, indicating a greater reliance on detailed reasoning cues over sheer quantity. This emphasises the importance of tailoring RAG strategies to specific model architectures and capabilities.

Chapter 6

Program-Aided Language Models

Program-Aided Language modelling represents a paradigm that incorporates symbolic program execution into natural language generation to replicate reasoning abilities for LLMs. Unlike direct text generation, where LLMs solely rely on training knowledge, PAL involves a hybrid between natural language and programmatic computation. The PAL methodology is built on the principle of generating a pseudocode or a program, which, on execution, produces a final answer. This enables the LLM to offload logically and computationally intensive tasks to an interpreter, improving accuracy in tasks requiring arithmetic and reducing runtime.

The strength of PAL lies in its separation of natural language understanding and execution-based reasoning. By allowing models to produce interpretable programs, PAL reduces the burden on the model to simulate complex steps and instead relies on program execution. This methodology can be combined with static prompts and dynamic few-shot examples, offering flexibility in adaptation to diverse task setups. Also, we have explored the implications of utilising a Python Reasoning Engine like Resource Description Framework to capture structured knowledge graphs for logic-based interpretation.

6.1 PAL from Scratch for Python Code

This section presents the performance of different LLMs using Program-Aided Language Models from scratch in plain Python code. The models are prompted to generate executable Python code to solve reasoning problems instead of natural language. Their performances were scored out of 100. The final scores are summarised in Table 6.1.

Model	**Score (/100)**
Gemini Flash 1.5	54
Llama-3.1 8B	32
Llama-3 70B	48
DeepSeek-LLM 7B	3
DeepSeek-coder 6.7B	20

Table 6.1: PAL from Scratch scores for different LLMs.

Some key observations can be drawn from the results. Firstly, the Python PAL approach shows a significant fall in performance from the direct response generation techniques. This indicates that code generation is not a strong forte for most renowned LLM models. Compared to their baseline performance, both Gemini Flash 1.5 and Llama-3 70B show decreased scores when using Python PAL (Gemini Flash drops from 78 to 54, Llama-3 70B from 55 to 48), indicating that code generation introduces additional complexity that challenges even sophisticated models. Llama-3.1 also exhibits a drop in performance between baseline (52) and Python PAL (32), displaying difficulties in code generation.

The Python PAL approach shows significant performance variation across different models. Gemini Flash 1.5 achieves the highest score, followed by Llama-3 70b and Llama-3.1 8B. There is a substantial gap between these models and DeepSeek-LLM, which scores only 3. The specialised DeepSeek-coder model outperforms the standard DeepSeek-LLM, demonstrating the importance of code-specific training for Python-based reasoning tasks. This suggests that models trained specifically for code generation have an advantage in PAL implementations. In addition, the DeepSeek-LLM fails to reach logical conclusions even in coding tasks.

These results suggest that while Python PAL offers potential benefits for reasoning tasks, its effectiveness depends on the model's scale and specialised training.

6.2 PAL from Scratch for RDF Code

In this section, we replicate the previous methodology with one significant change. We focus on producing RDF reasoning engine code instead of plain Python code. We can see that the PAL approach using RDF code shows consistently poor performance across all tested models. It consistently underperformed compared to Python-based implementations across all models, suggesting that Python's more accessible syntax and broader utility in problem-solving make it a more suitable pathway for reasoning tasks.

Gemini Flash 1.5 achieves the highest score at just 16, while other models perform even worse. The specialised DeepSeek-coder model performs at the same level as Llama-3 (6), which is slightly better than the standard DeepSeek-LLM (3). Still, the improvement is minimal compared to what we observed with Python-based PAL. The final scores are summarised in Table 6.2.

Model	**Score (/100)**
Gemini Flash 1.5	16
Llama-3.1 8B	7
Llama-3 70B	6
DeepSeek-LLM 7B	3
DeepSeek-coder 6.7B	6

Table 6.2: PAL from Scratch (RDF) scores for different LLMs.

The generally poor performance of RDF-based PAL across all models (with scores ranging from 3 to 16) indicates that RDF's more specialised syntax poses significant challenges for current LLMs. Even Gemini Flash 1.5, which performed reasonably well with Python PAL, struggled considerably with RDF generation.

These results suggest that while PAL approaches offer potential benefits for reasoning tasks, their effectiveness depends on the programming language used, the model's training focus, and overall scale. Python-based PAL shows more promise than RDF-based implementations, but neither approach consistently improves upon baseline performance across the evaluated models. This indicates that code generation and execution introduce opportunities and challenges for LLM reasoning, with task-specific training and larger model scales appearing to be important factors for success in this methodology.

6.3 PAL with Static One-Shot Prompting for RDF Code

This section examines the Program-Aided Language model approach with static one-shot prompting using RDF code syntax. Unlike the previous approach, this technique provides the models with a sample RDF code implementation for a similar task to serve as a template for the model to grasp the syntax and structure for RDF-based reasoning. Despite this scaffolding, models demonstrated difficulty with RDF's complex semantics and syntax.

The results show minimal improvement compared to the baseline RDF implementation, with Gemini Flash 1.5 achieving the highest score at 48. Despite being provided with RDF examples, Gemini consistently produced Python code instead of RDF code, an interesting failure mode suggesting that the model's training bias towards Python was strong enough to override the explicit RDF examples provided. The final scores are summarised in Table 6.3.

Model	**Score (/100)**
Gemini Flash 1.5	48*
Llama-3.1 7B	6
Llama-3 70B	1
DeepSeek-LLM 7B	6
DeepSeek-coder 6.7B	5

Table 6.3: PAL with static one-shot prompting (RDF) scores for different LLMs.

*Gemini Flash consistently produced plain Python code instead of RDF.

This technique's performance demonstrates that even with static examples, all models struggled to adapt to RDF's semantic triple structure and specialised syntax even with static examples. Despite the RDF example, Gemini Flash's consistent deviation from Python code highlights a significant challenge in this approach: models tend to default to more familiar programming paradigms when faced with less common formats like RDF. This phenomenon suggests that stronger interventions or more fine-tuned training would be needed for models to effectively utilise RDF for reasoning tasks.

6.4 PAL with Dynamic Few-Shot Prompting for Python Codes

This section examines the Program-Aided Language model approach with dynamic few-shot prompting using Python code. Unlike one-shot example prompting, this technique utilises a document as a knowledge corpus. It employs Retrieval-Augmented Generation to provide relevant programming examples as dynamic contexts for the model. This approach aims to enhance reasoning capabilities by retrieving problem-specific code patterns rather than relying on generic templates.

The results show varied performance across different models, with Gemini Flash 1.5 achieving the highest score at 55, representing one of the strongest performances in the Python-based PAL methodologies. Llama-3.1 8B scored 22, decreasing from its baseline Python implementation. Llama-3 70B performed notably poorly at just 7, a significant drop compared to its baseline Python implementation score. DeepSeek-LLM achieved 8, while the specialised DeepSeek-coder performed better at 15. The final scores are summarised in Table 6.4.

The dynamic few-shot approach produced mixed results compared to the baseline Python implementation. For Gemini Flash and DeepSeek models, the dynamic approach did not affect performance much, suggesting that retrieving relevant code examples provided marginal benefits for these models. However, Llama-3.1 8B showed a substantial decrease (22 vs 32),

Model	Score (/100)
Gemini Flash 1.5	55
Llama-3.1 8B	22
Llama-3 70B	7
DeepSeek-LLM 7B	8
DeepSeek-coder 6.7B	15

Table 6.4: PAL with dynamic few-shot prompting (Python) scores for different LLMs.

indicating that the dynamically retrieved examples may have introduced confusion rather than clarity.

Llama-3 70B's significant performance drop was most concerning, from 48 to just 7, suggesting that this model struggled considerably with integrating dynamically retrieved code examples. This dramatic reduction in performance highlights a potential vulnerability in larger models when faced with context that may contain irrelevant or conflicting information. Most of the code produced in this Llama-3 experiment had syntactic errors, which was not an issue when producing code from scratch. These results indicate that dynamic few-shot prompting with Python codes should offer potential benefits, but its effectiveness is not very explicit. This suggests that the quality and relevance of dynamically retrieved examples, rather than just the technique itself, may be crucial determinants of success in this approach.

6.5 PAL with Dynamic Few-Shot Prompting for RDF Codes

This section examines the Program-Aided Language model approach with dynamic few-shot prompting using RDF code. Similar to the Python-based dynamic approach, this technique utilises a knowledge corpus to retrieve relevant RDF code examples as context for solving reasoning problems. The strategy aims to overcome the challenges of RDF syntax by providing dynamically selected, problem-specific examples.

The results show consistently poor performance across all models, with Gemini Flash 1.5 achieving the highest nominal score at 47. However, it's important to note that despite being provided with dynamically retrieved RDF examples, Gemini consistently produced Python code instead of RDF code, a significant deviation highlighting the model's strong bias towards more familiar programming paradigms. It is to be noted that DeepSeek-LLM completely failed with 0, and DeepSeek-coder reached only 4. The final scores are summarised in Table 6.5.

Model	Score (/100)
Gemini Flash 1.5	47*
Llama-3.1 8B	19
Llama-3 70B	12
DeepSeek-LLM 7B	0
DeepSeek-coder 6.7B	4

Table 6.5: PAL with dynamic few-shot prompting (RDF) scores for different LLMs.

*Gemini Flash consistently produced plain Python code instead of RDF.

The dynamic approach showed mixed results compared to static example prompting with RDF. Llama-3.1 8B significantly improved its performance (19 vs 6). Llama-3 70B also showed improvement (12 vs 1), suggesting that dynamically retrieved examples were more helpful than static ones for these models when working with RDF syntax. However, DeepSeek-LLM declined from 6 to 0, while DeepSeek-coder remained similar (4 vs 5).

Notably, this methodology produced the highest scores for RDF implementation across all the approaches tested, though these scores remain far below those achieved with Python implementations. This suggests that while dynamic few-shot prompting offers some benefits for RDF code generation, the fundamental challenges of RDF syntax remain a significant barrier for current LLMs. Despite the dynamic retrieval of RDF examples, Gemini's persistent deviation to Python code confirms a strong training bias towards more common programming languages. This suggests that even advanced retrieval techniques cannot easily overcome deep-seated model preferences developed during training.

Chapter 7

Agentic Chain-of-Thought Methodologies

Chain-of-Thought prompting represents a paradigm that enables LLMs to break down complex reasoning tasks into sequential, interpretable steps rather than producing answers in a single pass. Unlike direct response generation, where LLMs may struggle with complex reasoning, CoT approaches guide models through intermediate reasoning states, enhancing performance on tasks requiring multi-step logical inference [22]. While traditional CoT relies heavily on human guidance, verifying each step in the reasoning chain, this dependency creates significant challenges in scalability and introduces potential biases.

The challenges of human-in-the-loop CoT are substantial. Human verification introduces the "Clever Hans" effect [7], where models appear to reason correctly but are guided towards known answers through iterative human feedback. This approach requires constant human attention, limiting throughput and introducing inconsistencies across different human verifiers. Moreover, when humans lack domain expertise to verify complex reasoning steps, the approach becomes fundamentally limited in its applicability to advanced problem domains.

Agentic CoT addresses these limitations by automating the reasoning verification process. Rather than relying on human intervention, agentic approaches empower LLMs to verify their reasoning through structured tools and feedback mechanisms [23]. This methodology can be implemented with or without internet access, creating different capabilities and constraints. Internet-enabled agentic CoT can leverage vast external knowledge to supplement reasoning chains, while internet-free implementations must rely solely on internal knowledge representation and self-verification.

In our experimental evaluation, we conduct several key investigations: comparing internet-enabled versus internet-free agentic CoT implementations and evaluating performance differences across model architectures (including OpenHermes and Gemma-2 variants). These experiments aim to quantify the benefits of agentic approaches over traditional methods and identify optimal configurations for different reasoning tasks and model architectures.

We exclude models such as Llama-3.1 and DeepSeek-LLM from the internet-enabled agentic CoT experiments due to known compatibility issues with CrewAI's web browsing tools. As noted by the CrewAI development team in an official GitHub comment on Issue [24], smaller open-source LLMs frequently fail when required to follow complex tool-calling schemas during web interaction. These failures manifest as repeated "Invalid Format: Missing 'Action' after 'Thought'" errors, indicating a struggle to maintain structured reasoning output across multiple tool invocations. While specialised models like OpenAI GPT-4 and Claude perform reliably in such contexts, open-source models like Llama and DeepSeek are less robust due to shorter context windows and limited instruction-following capabilities. Thus, our choice to omit them from these experiments ensures the stability and interpretability of our evaluation results.

7.1 Agentic CoT Methodologies Without Internet

This section presents the performance of different large language models using the Agentic Chain-of-Thought methodology without internet access. In this approach, models are prompted to break down reasoning tasks into sequential steps and verify their reasoning without accessing external knowledge sources. Their performances were scored out of 100. The final scores are summarised in Table 7.1.

Model	Score (/100)
OpenHermes 7B	18
Gemma-2 9B	43

Table 7.1: Scores for different LLMs using CoT without internet.

Some key observations can be drawn from the results. The Agentic CoT approach without internet access shows interesting performance variations across the tested models. Most notably, Gemma-2 9b achieved a significantly higher score (43) than OpenHermes 7b (18), suggesting that model architecture and parameter count substantially impact self-verification capabilities when external knowledge is unavailable.

The performance of these frameworks is poorer than that of most baseline methodologies, which is not exactly desirable considering the significant time consumed to unfold conversations between agents. Hallucinations in CoT conversations were prominent at transition points between reasoning steps, where models fabricate connections to maintain logical flow. The step-by-step nature of CoT can make these hallucinations more convincing, as the structured format gives an appearance of methodical verification even when facts are incorrect. This risk is particularly pronounced in specialised domains where the model's training data may be sparse. Unlike internet-enabled approaches, where hallucinations can be fact-checked against external sources, offline CoT relies entirely on the model's internal knowledge representation, which may contain outdated or incorrect information.

Despite these challenges, the offline CoT approach offers practical advantages when internet access is limited, privacy concerns exist, or consistent reasoning independent of external data is required.

7.2 Agentic CoT Methodologies With Internet

This section examines the performance of different large language models using the Agentic Chain-of-Thought methodology with internet access. In this approach, models are prompted to break down reasoning tasks into sequential steps while being able to search for and incorporate external information from the web. Their performances were scored out of 100. The final scores are summarised in Table 7.2.

Model	Score (/100)
OpenHermes 7B	14
Gemma-2 9B	29

Table 7.2: Scores for different LLMs using CoT with internet.

Surprisingly, the internet-enabled Agentic CoT approach consistently underperformed compared to its internet-free counterpart across both tested models. The performance gap between models remains consistent with the internet-free approach, with Gemma-2 9B (29) significantly outperforming OpenHermes 7B (14). This suggests that model architecture and parameter count impact reasoning capabilities and the ability to integrate external information effectively. OpenHermes 7B scored only 5 with internet access, substantially lower than its score without internet(14). Similarly, Gemma-2 9B achieved just 29 with internet, compared to 43 without internet access.

This counter-intuitive result challenges the assumption that more information leads to better reasoning. The performance degradation likely stems from an information overload problem in the agentic system. When models can access vast external knowledge sources, they face several challenges. The models struggle to distinguish between relevant and irrelevant information retrieved from the web, potentially incorporating misleading or tangential data into their reasoning chains. Also, when faced with potentially conflicting information between their parametric knowledge and retrieved data, models must resolve these contradictions, which appears particularly challenging for current architectures.

Despite the theoretical advantages of internet-enabled reasoning, these results highlight that current models face significant challenges in effectively utilising web-based information within reasoning chains. As agentic frameworks evolve, developing better mechanisms for information filtering, relevance assessment, and contradiction resolution will be crucial for realising the potential benefits of internet-augmented reasoning.

Chapter 8

Analysis

This chapter comprehensively analyses the performance of the Large Language Models across multiple reasoning enhancement methodologies and model architectures. Based on the individual experiments detailed in the preceding chapters, we now delve into the underlying patterns and provide insights about the different techniques and model families.

The comparison of performances of all methodologies across models is visually summarised in Figure 8.1.

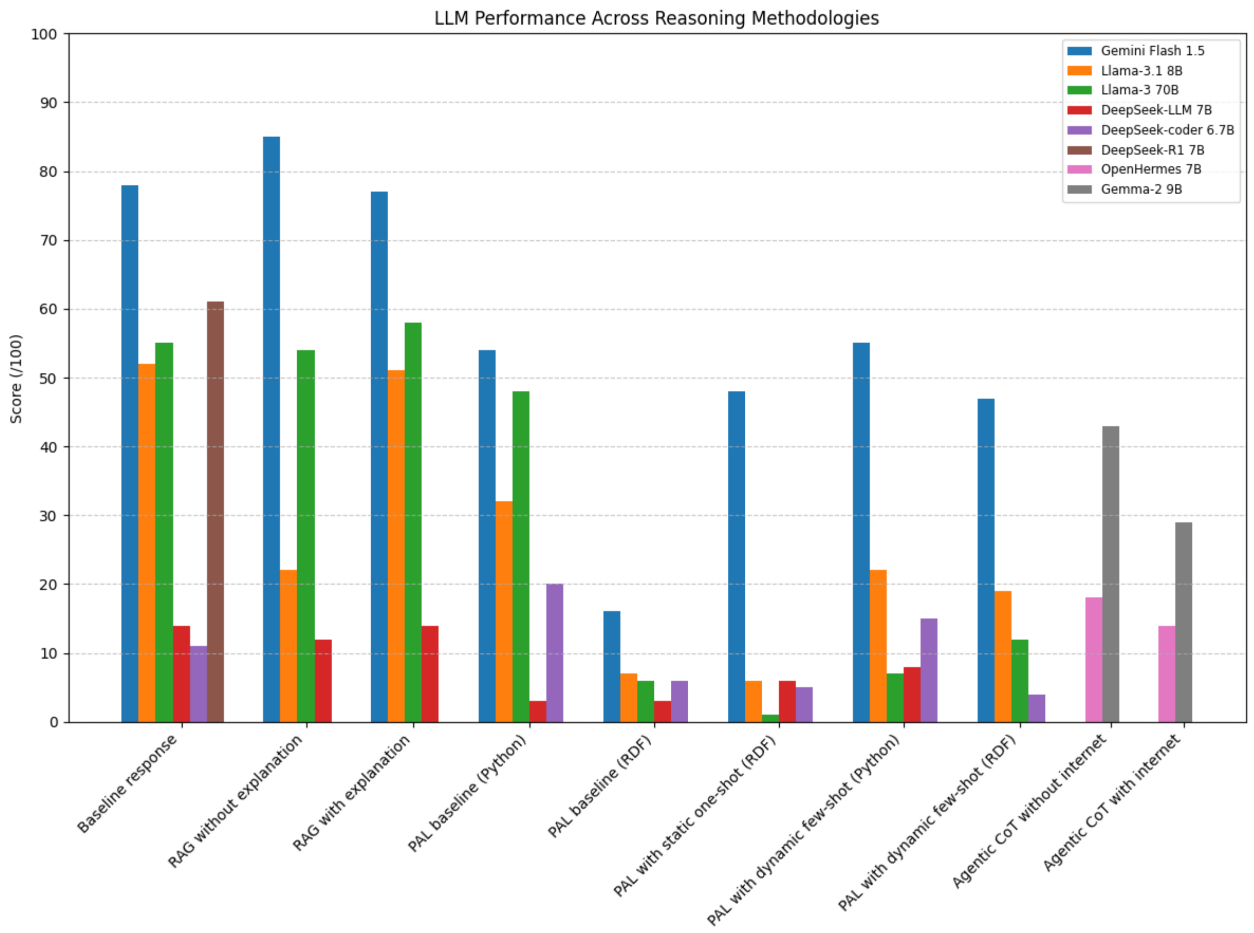


Figure 8.1: Performance comparison across different LLMs.

The visualisation in Figure 8.1 provides a multi-dimensional view of model performance across the different experimental frameworks, revealing significant variations requiring more investigation.

The analysis follows a structured approach that examines performance across three critical dimensions. First, we benchmark methodology effectiveness, comparing RAG implementations with and without explanations, static one-shot vs dynamic few-shot prompting approaches, plain Python versus RDF code generation in PAL frameworks, and the impact of internet access on Agentic Chain-of-Thought reasoning. Second, we investigate architectural differences, contrasting open-source models with closed-source alternatives, analysing the effects of parameter scaling, and assessing how specialised training (for code generation, reasoning, or general language tasks) influences performance on reasoning challenges. Finally, we evaluate generalisation capabilities through performance consistency across multiple runs, identifying which models demonstrate reliable reasoning versus those exhibiting high variability.

This multifaceted evaluation framework better fits current emerging best practices in LLM benchmarking, emphasising the necessity of rating model performance across diverse tasks rather than relying on single-metric comparisons[25]. As noted in recent literature on LLM evaluation, comprehensive assessment requires examining qualitative aspects like reasoning coherence and multi-hop capabilities, as well as quantitative aspects like accuracy, inference latency, etc.

This analysis aims to provide actionable insights for selecting optimal configurations based on specific application requirements and computational constraints by systematically exploring the interplay between model architectures and reasoning methodologies.

The findings presented in this chapter highlight the current capabilities and limitations of various LLM reasoning approaches and point towards promising directions for future development in enhancing LLMs' logical inference capabilities. Through this comparative analysis, we seek to move beyond anecdotal observations towards a more rigorous understanding of the factors that govern reasoning performance in large language models.

8.1 Results

This section presents the comprehensive findings from our evaluation of LLM reasoning capabilities across multiple methodologies, model architectures, and training approaches. Our experiments generated a substantial dataset comparing ten distinct reasoning enhancement techniques across different models, providing insights into the strengths and limitations of current approaches to improving logical reasoning in large language models.

Our analysis is structured along three primary dimensions: methodology benchmarking, where we compare different reasoning enhancement approaches; architectural comparisons, examining how model design and training influence reasoning capabilities; and generalisation assessment, investigating performance consistency across multiple evaluation runs. Through this multifaceted examination, we identify common patterns and unexpected findings that extend our understanding of how to enhance reasoning capabilities in large language models effectively.

8.1.1 Methodology Benchmarking

8.1.1.1 RAG with explanation vs RAG without explanation

The results of Retrieval-Augmented Generation with and without explanation context revealed significant variations in effectiveness across different model architectures. Examining the performance data, we can observe that RAG approaches provided inconsistent results compared to baseline performance, with most improvements being marginal or even decremental, except for Gemini.

Gemini Flash 1.5 showed remarkable improvement using RAG without explanation (85% compared to a baseline of 78%), but declined when RAG with explanation was used (77%). This suggests that exposure to a larger number of examples provides more benefit than fewer, deeply explained examples for advanced closed-source models.

Conversely, Llama-3.1 8B showed substantially better performance using RAG with explanation (51%) than without explanation (22%), though both approaches underperformed compared to their baseline (52%). This pattern indicates that open-source models with lower parameter counts benefit more from explicit reasoning paths than from mere exposure to similar problems. Llama-3 70B showed minimal differences between the approaches, with a slight preference for explained examples (58% vs 54%). DeepSeek-LLM 7B showed only marginal differences between the two RAG approaches (14% vs 12%), with both performing near or slightly below their baseline, suggesting that RAG provided limited benefits for this model architecture.

These findings highlight that RAG's effectiveness is highly model-dependent, with the observed differences stemming from the trade-off between quantity and depth of examples. The results suggest that closed-source models might benefit more from exposure to a larger corpus of questions. In contrast, open-source models gain more from fewer, thoroughly explained examples, indicating a greater reliance on detailed reasoning cues over sheer quantity. This emphasises the importance of tailoring RAG strategies to specific model architectures and capabilities rather than applying a one-size-fits-all approach.

8.1.1.2 Static One-Shot vs Dynamic Few-shot Adaptation

We compared static prompting with dynamic few-shot adaptation to assess the adaptability of each method in handling reasoning tasks. This comparison revealed notable differences in model performance, particularly in the context of RDF code generation. With static examples, most models demonstrated minimal improvement over baseline implementations, with some models even showing performance degradation. Llama-3 70B experienced a decrease from 6% with an RDF baseline implementation to just 1% with static examples, suggesting that fixed examples introduced confusion rather than clarity. Gemini Flash 1.5 was an exception, showing improvement with static examples (48%), though it consistently produced Python code instead of the requested RDF format.

Dynamic few-shot adaptation yielded significantly better results for mid-tier models. Llama-3.1 8B showed a remarkable improvement from 6% with static prompting to 19% with dynamic few-shot adaptation, representing more than a threefold increase in performance. Similarly, Llama-3 70B improved from 1% to 12% when switching from static to dynamic examples. This pattern suggests that contextually relevant examples retrieved dynamically are particularly beneficial for models with moderate reasoning capabilities. However, the benefits were not universal. DeepSeek-LLM's performance dropped from 6% to 0%, indicating that dynamic adaptation may introduce additional complexity that overwhelms models with weaker baseline reasoning abilities. These results highlight the importance of matching prompting strategies to model capabilities when implementing RDF-based reasoning frameworks.

8.1.1.3 PAL for Python Code Generation vs PAL for Reasoning Engine Frameworks

Our comparison of Python-based and RDF-based Program-Aided Language Models revealed a significant performance disparity across all evaluated models. Python PAL implementations consistently outperformed their RDF counterparts, with Gemini Flash 1.5 achieving the highest scores in both categories (54% for Python vs. 16% for RDF from scratch). When provided with RDF examples or templates, Gemini consistently reverted to generating Python code instead, indicating a strong training bias towards familiar programming paradigms. The most striking gap appeared with Llama-3 70B, which scored 48% with Python but only 6% with RDF.

Advanced prompting techniques yielded mixed improvements but maintained the advantage for Python. Dynamic few-shot prompting marginally enhanced Python implementation for Gemini (55%) and DeepSeek-LLM (8%), while significantly improving RDF performance for some models (Llama-3.1 8B increased from 7% to 19%). However, this approach dramatically reduced Llama-3 70B's Python performance (dropping from 48% to 7%), suggesting that dynamic retrieval may sometimes introduce disruptive context. Despite these variations, all models struggled with RDF's specialised syntax and semantic structure. These results demonstrate that while RDF frameworks theoretically offer advantages for structured relational reasoning, current LLMs lack sufficient exposure to RDF patterns for effective code generation in this paradigm.

8.1.1.4 Agentic CoT Prompting with and without Internet Access

Our comparison of Agentic Chain-of-Thought methodologies with and without internet access revealed a surprising pattern contrary to initial expectations. Across both tested models, performance was consistently better when internet access was disabled. Gemma-2 9B achieved a significantly higher score without internet (43%) compared to with internet access (29%), representing a substantial performance decrease when web retrieval was enabled. Similarly, OpenHermes 7B scored 18% without internet versus only 14% with internet access.

This counter-intuitive finding suggests that internet access created an "information overload" problem for agentic systems. When provided with the ability to search and retrieve external information, models struggled to distinguish between relevant and irrelevant content, often incorporating tangential or misleading data into their reasoning chains. Resolving contradictions between the models' parametric knowledge and newly retrieved information was particularly challenging. The hallucination patterns also differed between conditions, with internet-enabled models producing more varied factual errors influenced by retrieved content. In contrast, offline models exhibited more predictable hallucinations at transition points between reasoning steps, where they fabricated connections to maintain logical flow. Despite lacking real-time information access, the offline approach provided more coherent reasoning paths by forcing models to rely on internal knowledge representation rather than potentially distracting external sources.

8.1.2 Architectural Comparisons

8.1.2.1 Open-source vs Closed-source Models

Our analysis of model performance across reasoning methodologies demonstrates the performance gap between closed-source and open-source models. Gemini Flash 1.5 consistently outperformed open-source alternatives across nearly all evaluation approaches. For example, Gemini scored 78% on baseline reasoning tasks compared to Llama-3.1 8B's 55% and DeepSeek-LLM's 15%. This performance gap was particularly pronounced in direct reasoning tasks, but narrowed somewhat when enhancement techniques like RAG with explanation were applied. These results align with broader industry benchmarks showing closed-source models generally outperforming open-source alternatives on reasoning tasks. However, the gap is narrowing with larger parameter models and specialised training techniques.

8.1.2.2 Parameter Scaling Effects

Our analysis of parameter scaling effects between Llama-3.1 8B and Llama-3 70B models revealed significant performance variations across reasoning methodologies. While baseline performance was surprisingly comparable, methodology-specific differences highlighted the advantages of larger parameter counts. The 70B model demonstrated superior performance in Python code generation (48% vs. 32% for the 8B model) and maintained consistent performance across different RAG implementations. Most notably, when using RAG without explanation, the 70B model maintained strong performance (54%). In comparison, the 8B model declined dramatically (22%), suggesting the larger model's enhanced ability to identify relevant patterns without explicit guidance.

The 70B model's advantages align with its benchmark performance on standard evaluations, where it consistently outperforms the 8B variant across reasoning-intensive tasks. However, our experiments revealed an interesting exception: both models struggled similarly with RDF code generation. This unexpected result suggests that parameter scaling alone doesn't necessarily improve performance on unfamiliar tasks where neither model has sufficient training exposure.

8.1.2.3 Specialised Training Effects

Our comparison of DeepSeek model variants with different training specialisations revealed dramatic performance differences despite similar parameter counts. DeepSeek-R1 7B, trained explicitly for reasoning through reinforcement learning, achieved an impressive 61% on baseline reasoning tasks, more than four times higher than DeepSeek-LLM 7B (14%) despite having comparable parameter counts. This substantial gap demonstrates how specialised reasoning training can produce capabilities far beyond what general language model training provides alone. DeepSeek-coder 6.7B, explicitly trained for code generation, scored 11% on general reasoning tasks, below even the general-purpose DeepSeek-LLM, suggesting that code specialisation alone doesn't necessarily transfer to improved reasoning on non-coding problems. While DeepSeek-R1 showed remarkable performance on baseline tasks, we couldn't implement it across our complete methodology set due to its long inference time of approximately 25 minutes per problem. This significant runtime constraint highlights a vital trade-off in specialised reasoning models: superior performance often comes at the cost of computational efficiency. The stark performance differences between these similarly-sized models underscore how targeted training objectives can dramatically shape a model's reasoning capabilities, with DeepSeek-R1's reinforcement learning approach demonstrating particular promise for enhancing logical inference abilities.

8.1.3 Generalisation Assessment

8.1.3.1 Performance Drift Across Multiple Runs

Below, we present the performance scores of various LLMs under the baseline evaluation methodology across two independent runs.

Model	**Run 1 (/100)**	**Run 2 (/100)**
Gemini Flash 1.5	78	81
Llama-3.1 8B	55	52
Llama-3 70B (hosted on Groq)	55	55
DeepSeek-LLM 7B	15	14
DeepSeek-coder 6.7B	11	12

Table 8.1: Scores from two baseline runs of different LLMs.

The table below details the consistency of each model's responses across the two runs. These statistics highlight varying degrees of performance drift and reasoning stability among the models:

Model	**Correct Twice**	**Correct Once**	**Incorrect Twice**
Gemini Flash 1.5	74	11	15
Llama-3.1 8B	36	35	29
Llama-3 70B (hosted on Groq)	55	0	45
DeepSeek-LLM 7B	73	25	2
DeepSeek-coder 6.7B	5	23	72

Table 8.2: Evaluation consistency across two runs.

Despite similar overall scores, significant differences in run-to-run consistency highlight the impact of model architecture and deployment platform on reasoning stability. Gemini Flash 1.5 shows high performance with only 11 mismatches, indicating 89% consistency and good reliability in maintaining correct predictions. Llama-3 70B hosted on Groq exhibits perfect consistency with zero mismatches, emphasising the benefits of parameter scaling. In contrast, Llama-3.1 8B shows the poorest stability with 35 mismatches and substantial prediction changes, underscoring the instability of smaller models. Both DeepSeek-LLM 7B and DeepSeek-coder 6.7B demonstrate moderate consistency but show a strong bias towards negative predictions. These results confirm that larger models offer more dependable performance across repeated evaluations, an essential factor in applications demanding consistent reasoning.

Chapter 9

Applications of LLM reasoning capabilities

This chapter explores practical applications of the reasoning enhancement techniques evaluated in our research. The findings from our comparative analysis reveal that different reasoning methodologies are suited to different types of applications, with trade-offs between performance, computational efficiency, and domain specificity. We list several high-impact application areas where LLM reasoning capabilities can address complex challenges across various domains.

9.1 Complex Problem Solving

9.1.1 Mathematical Problem Solving

Our findings on PAL implementations reveal particular promise for mathematical problem solving. By converting natural language problems into executable code, models can leverage computational precision for probability calculations, algebraic manipulations, and geometric reasoning. This approach mirrors recent developments in specialised reasoning models that have achieved breakthrough performance on advanced mathematical benchmarks[26].

9.1.2 Multi-hop Query Resolution

LLM reasoning capabilities enable effective handling of multi-hop query questions requiring reasoning over multiple pieces of information. As noted in recent research, multi-hop queries (questions requiring reasoning over numerous pieces of information) and multifaceted queries (broad queries with multiple sub-intents or aspects) present significant challenges for traditional search systems[27]. Our evaluation of RAG with explanation approaches demonstrates how retrieval-enhanced reasoning can connect disparate pieces of information to form coherent reasoning chains. Studies have developed different techniques that tightly integrate Knowledge Graph structures and semantics into LLM representations.

9.2 Knowledge Management and Information Processing

9.2.1 Document Analysis and Synthesis

In legal contexts, reasoning-enhanced LLMs can absorb a great deal of information when writing internal memoranda and produce papers for arbitration or litigation, which can serve as a strong base for developing and refining legal arguments [28]. Our findings on RAG implementation effectiveness are particularly relevant here, highlighting how models can integrate retrieved information into coherent analyses.

9.2.2 Repository Navigation and Database Reasoning

The ability to reason over interlinked information is crucial for knowledge management systems. Our evaluation of agentic frameworks demonstrates how models can maintain coherent reasoning while navigating complex information structures. This capability enables applications to handle issues on repositories with interlinked files parsed as a database, as mentioned in emerging applications of graph-based retrieval methods[27].

9.3 Automation and Robotics

9.3.1 Enhanced Robotic Perception and Navigation

LLMs with reasoning capabilities offer significant advantages for robotics, particularly in perception, decision-making, control, and interaction. Structured reasoning approaches like RDF-based PAL highlight potential pathways for integrating semantic understanding with physical navigation tasks. As noted in robotics research, LLMs enable knowledge acquisition and reasoning capabilities to help robots acquire and process rich multimodal knowledge and improve decision-making ability and intelligence[29].

9.3.2 Optimised Path Planning

The integration of reasoning capabilities with sensory data enables robots to determine efficient traversal paths based on environmental constraints with limited/active sensing. The flexibility and adaptability of reasoning-enhanced LLMs would allow robots to adapt to different tasks and environments and make flexible adjustments and self-adaptations based on specific circumstances[29].

9.4 Education and Learning

9.4.1 Virtual Tutoring

Reasoning-enhanced LLMs demonstrate significant potential for educational applications. The performance variations observed across different reasoning enhancement techniques in our study highlight the importance of matching methodologies to specific academic contexts. The ability to break down complex concepts into step-by-step explanations, as evaluated in our Chain-of-Thought experiments, enables the creation of personalised learning experiences. By decomposing topics based on individual student needs, reasoning-enhanced LLMs can support children's education, language learning, etc. Recent advancements in reasoning LLMs have made them particularly valuable for education and entertainment purposes, where they can provide educational content, answer questions, or engage in games and entertainment activities[30].

9.5 Professional Services

9.5.1 Customer Service Automation

Our evaluation of agentic reasoning frameworks is directly relevant to customer service applications. Recent literature notes that reasoning LLMs are particularly valuable when integrating them as multi-agent models for seamless question-and-answer in customer support-based AI chatbots[31]. The ability to maintain reasoning coherence across multiple interactions enables more effective resolution of complex customer inquiries.

9.5.2 Event Planning and Resource Optimisation

The problem-solving capabilities evaluated in our research enable sophisticated planning applications that require adherence to multiple constraints. Event planning represents an ideal application area where reasoning-enhanced LLMs can design optimised schedules, detailed plans, and efficient event timelines, ensuring resource utilisation and adherence to constraints[32].

9.5.3 Mechanical Failure Analysis

Industrial applications benefit from the ability to reason across multiple data sources. By analysing failure reports and sensor data to identify root causes, reasoning-enhanced LLMs can provide actionable insights for mitigating issues in mechanical systems or industrial operations. The feature generation capabilities of LLMs, as noted in recent research, can be particularly valuable for extracting insights from complex industrial data[33].

9.6 Conclusion

The reasoning enhancement techniques evaluated in our research demonstrate significant potential across diverse application domains. Our findings suggest that application-specific optimisation is crucial, as different reasoning techniques may be optimal depending on the application context. As reasoning capabilities in LLMs continue to evolve, we anticipate increasing adoption across these application areas, with particular growth in domains requiring multi-step problem solving and integrating diverse information sources.

Chapter 10

Future Work

This thesis explores several methodologies for enhancing reasoning capabilities in Large Language Models (LLMs), but numerous promising directions remain for future investigation. The following areas represent valuable opportunities for extending this research.

10.1 Expanding Model Diversity

Our current evaluation was constrained by computational limitations and platform restrictions. Future work could incorporate frontier reasoning-specialised models such as OpenAI's `o1` (which achieves 90% accuracy on AIME benchmarks)[34], GPT-4o (which excels in versatility, though has limitations in step-by-step logical tasks), and Qwen QwQ (which scored 90.6% on MATH-500)[35]. Including Claude 3.7 Sonnet, Anthropic's first hybrid reasoning model released in February 2025, and Mistral Large would provide more comprehensive benchmarking across model architectures and training approaches.

10.2 Parameter Scaling Analysis

Our experiments primarily utilised models with parameter counts below 10 billion due to resource constraints. A systematic evaluation of how reasoning capabilities scale with parameter count could reveal important insights about the relationship between model size and reasoning quality. As noted in research on frontier models, the performance of reasoning models consistently improves with more reinforcement learning (train-time compute) and with more time spent thinking (test-time compute)[36]. Extending our methodology to larger models (70B+) would require dedicated computational infrastructure but could potentially reveal important pivot points in reasoning capabilities.

10.3 Advanced Agentic Frameworks

While our research implemented basic agentic reasoning approaches, future work could explore more sophisticated agent coordination strategies. Implementing the complete think-act-observe loop with specialised agents for different reasoning subtasks could significantly enhance performance. Future implementations could include dedicated information retrieval agents, computational agents for handling quantitative modelling, and conceptual mapping agents for organising ideas and dependencies. Controlling information flow between these specialised agents while maintaining reasoning coherence represents a significant research opportunity.

10.4 Graph-Enhanced Retrieval

GraphRAG represents a promising advancement over traditional RAG approaches. By leveraging knowledge graphs rather than relying solely on vector similarity, GraphRAG has demonstrated improvements in question-and-answer performance when conducting document analysis of complex information[37]. Future work could implement and evaluate GraphRAG architectures, which have shown up to 35% improvement in answer precision compared to vector-only retrieval methods. This approach would be particularly valuable for questions requiring traversal of disparate pieces of information connected through shared attributes.

10.5 Reproducibility Assessment

Our current evaluation included limited reproducibility testing. Future work could systematically analyse performance drift across multiple runs for all methodologies. As highlighted in recent research, for humans to be able to trust LLM-based applications, their outputs should be consistent when prompted with inputs that carry the same meaning or intent[38]. Implementing robust reproducibility testing would provide valuable insights into which reasoning enhancement techniques produce the most consistent results.

10.6 Emerging Techniques

Several recent developments warrant investigation. These include Chain of Guidance (CoG)[38], a multi-step prompting technique that enhances consistency, and specialised length reward mechanisms to address the overthinking problem in reasoning models. Additionally, new benchmarks like MMLU-Pro and GPQA Diamond provide more rigorous evaluation frameworks for advanced reasoning capabilities and could be incorporated into future testing protocols.

These research directions would build upon our current findings to develop more robust and reliable reasoning enhancement techniques for large language models, ultimately leading to AI systems with stronger logical inference capabilities across diverse application domains.

References

[1] S. Kambhampati, K. Stechly, and K. Valmeekam, "(How) Do reasoning models reason?," *Annals of the New York Academy of Sciences*, Apr. 2025. URL: http://dx.doi.org/10.1111/nyas.15339.

[2] Y. Wu, F. Jia, S. Zhang, H. Li, E. Zhu, Y. Wang, Y. T. Lee, R. Peng, Q. Wu, and C. Wang, "MathChat: Converse to Tackle Challenging Math Problems with LLM Agents," 2024. URL: https://arxiv.org/abs/2306.01337.

[3] S. Kambhampati, K. Valmeekam, L. Guan, M. Verma, K. Stechly, S. Bhambri, L. Saldyt, and A. Murthy, "LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks," 2024. URL: https://arxiv.org/abs/2402.01817.

[4] X. Wang, S. Tan, M. Jin, W. Y. Wang, R. Panda, and Y. Shen, "Do Larger Language Models Imply Better Reasoning? A Pretraining Scaling Law for Reasoning," 2025. URL: https://arxiv.org/abs/2504.03635.

[5] D. Tomaszuk and D. Hyland-Wood, "RDF 1.1: Knowledge Representation and Data Integration Language for the Web," *Symmetry*, vol. 12, p. 84, Jan. 2020. URL: http://dx.doi.org/10.3390/sym12010084.

[6] K. Valmeekam, S. Sreedharan, M. Marquez, A. Olmo, and S. Kambhampati, "On the Planning Abilities of Large Language Models (A Critical Investigation with a Proposed Benchmark)," 2023. URL: https://arxiv.org/abs/2302.06706.

[7] S. Kambhampati, "Can large language models reason and plan?," *Annals of the New York Academy of Sciences*, vol. 1534, p. 15–18, Mar. 2024. URL: https://dx.doi.org/10.1111/nyas.15125.

[8] A. Patil, "Advancing Reasoning in Large Language Models: Promising Methods and Approaches," 2025. URL: https://arxiv.org/abs/2502.03671.

[9] J. Boye and B. Moell, "Large Language Models and Mathematical Reasoning Failures," 2025. URL: https://arxiv.org/abs/2502.11574.

[10] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, "Retrieval-Augmented Generation for Large Language Models: A Survey," 2024. URL: https://arxiv.org/abs/2312.10997.

[11] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, and J. Larson, "From Local to Global: A Graph RAG Approach to Query-Focused Summarization," 2024. URL: https://arxiv.org/abs/2404.16130.

[12] L. Gao, A. Madaan, S. Zhou, U. Alon, P. Liu, Y. Yang, J. Callan, and G. Neubig, "PAL: Program-aided Language Models," 2023. URL: https://arxiv.org/abs/2211.10435.

[13] H. Chae, Y. Kim, S. Kim, K. T. iunn Ong, B. woo Kwak, M. Kim, S. Kim, T. Kwon, J. Chung, Y. Yu, and J. Yeo, "Language Models as Compilers: Simulating Pseudocode Execution Improves Algorithmic Reasoning in Language Models," 2024. URL: https://arxiv.org/abs/2404.02575.

[14] W. Chen, X. Ma, X. Wang, and W. W. Cohen, "Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks," 2023. URL: https://arxiv.org/abs/2211.12588.

[15] CrewAI Contributors, "CrewAI: Open-Source Multi-Agent Framework," 2024. URL: https://www.crewai.com/open-source.

[16] S. Hu, C. Lu, and J. Clune, "Automated Design of Agentic Systems," 2025. URL: https://arxiv.org/abs/2408.08435.

[17] H. Veeraboina, "AIME Problem Set (1983-2024)," 2024. URL: https://www.kaggle.com/datasets/hemishveeraboina/aime-problem-set-1983-2024.

[18] VALS.ai, "MATH500 Benchmark (05-05-2025)," 2025. URL: https://www.vals.ai/benchmarks/math500-05-05-2025.

[19] DeepSeek-AI, D. Guo, D. Yang, H. Zhang, *et al.*, "DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning," 2025. URL: https://arxiv.org/abs/2501.12948.

[20] Indian Institutes of Management, "Common Admission Test (CAT) Official Portal," 2025. URL: https://iimcat.ac.in/per/g06/pub/32842/ASM/WebPortal/1/index.html?32842@@1@@1.

[21] H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe, "Let's Verify Step by Step," 2023. URL: https://arxiv.org/abs/2305.20050.

[22] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," 2023. URL: https://arxiv.org/abs/2201.11903.

[23] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan, "Tree of Thoughts: Deliberate Problem Solving with Large Language Models," 2023. URL: https://arxiv.org/abs/2305.10601.

[24] joaomdmoura, "Comment on: crewaiinc/crewai issue #103," 2024. URL: https://github.com/crewAIInc/crewAI/issues/103#issuecomment-1902667402.

[25] Z. Guo, R. Jin, C. Liu, Y. Huang, D. Shi, Supryadi, L. Yu, Y. Liu, J. Li, B. Xiong, and D. Xiong, "Evaluating Large Language Models: A Comprehensive Survey," 2023. URL: https://arxiv.org/abs/2310.19736.

[26] C. Liu and R. Jabbarvand, "A Tool for In-depth Analysis of Code Execution Reasoning of Large Language Models," 2025. URL: https://arxiv.org/abs/2501.18482.

[27] X. Wu and K. Tsioutsiouliklis, "Thinking with Knowledge Graphs: Enhancing LLM Reasoning Through Structured Data," 2024. URL: https://arxiv.org/abs/2412.10654.

[28] S. S. Tu, A. Cyphert, and S. Perl, "Artificial Intelligence: Legal Reasoning, Legal Research and Legal Writing," 2024. Minnesota Journal of Law, Science and Technology, Volume 25, 105–125 (2024), WVU College of Law Research Paper No. 2024-012, URL: https://ssrn.com/abstract=4817765.

[29] F. Zeng, W. Gan, Y. Wang, N. Liu, and P. S. Yu, "Large Language Models for Robotics: A Survey," 2023. URL: https://arxiv.org/abs/2311.07226.

[30] J. Yu, Z. Zhang, D. Zhang-li, S. Tu, Z. Hao, R. M. Li, H. Li, Y. Wang, H. Li, L. Gong, J. Cao, J. Lin, J. Zhou, F. Qin, H. Wang, J. Jiang, L. Deng, Y. Zhan, C. Xiao, X. Dai, X. Yan, N. Lin, N. Zhang, R. Ni, Y. Dang, L. Hou, Y. Zhang, X. Han, M. Li, J. Li, Z. Liu, H. Liu, and M. Sun, "From MOOC to MAIC: Reshaping Online Teaching and Learning through LLM-driven Agents," 2024. URL: https://arxiv.org/abs/2409.03512.

[31] S. K. Dam, C. S. Hong, Y. Qiao, and C. Zhang, "A Complete Survey on LLM-based AI Chatbots," 2024. URL: https://arxiv.org/abs/2406.16937.

[32] C. H. Song, J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, and Y. Su, "LLM-Planner: Few-Shot Grounded Planning for Embodied Agents with Large Language Models," 2023. URL: https://arxiv.org/abs/2212.04088.

[33] D. Chandra, "Applications of Large Language Model Reasoning in Feature Generation," 2025. URL: https://arxiv.org/abs/2503.11989.

[34] S. Wu, Z. Peng, X. Du, T. Zheng, M. Liu, J. Wu, J. Ma, Y. Li, J. Yang, W. Zhou, Q. Lin, J. Zhao, Z. Zhang, W. Huang, G. Zhang, C. Lin, and J. H. Liu, "A Comparative Study on Reasoning Patterns of OpenAI's o1 Model," 2024. URL: https://arxiv.org/abs/2410.13639.

[35] Qwen Team, "QwQ: Reflect Deeply on the Boundaries of the Unknown." URL: https://qwenlm.github.io/blog/qwq-32b-preview/, November 2024.

[36] Z.-Z. Li, D. Zhang, M.-L. Zhang, J. Zhang, Z. Liu, Y. Yao, H. Xu, J. Zheng, P.-J. Wang, X. Chen, Y. Zhang, F. Yin, J. Dong, Z. Li, B.-L. Bi, L.-R. Mei, J. Fang, Z. Guo, L. Song, and C.-L. Liu, "From System 1 to System 2: A Survey of Reasoning Large Language Models," 2025. URL: https://arxiv.org/abs/2502.17419.

[37] H. Han, H. Shomer, Y. Wang, Y. Lei, K. Guo, Z. Hua, B. Long, H. Liu, and J. Tang, "RAG vs. GraphRAG: A Systematic Evaluation and Key Insights," 2025. URL: https://arxiv.org/abs/2502.11371.

[38] H. Raj, V. Gupta, D. Rosati, and S. Majumdar, "Improving Consistency in Large Language Models through Chain of Guidance," 2025. URL: https://arxiv.org/abs/2502.15924.