Exploring Reasoning Abilities of Large Language Models

Pranoy K P ED20B045
Guides: Prof. Sivaram Ambikasaran, Dr. Sri Vallabha Deevi
Indian Institute of Technology Madras

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

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. LLMs can be fine-tuned for specific tasks or guided by prompt engineering. However, their ability to reason and plan remains an open research challenge. This project uses a structured evaluation across various tasks to investigate 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 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. This project helps to understand the practical limitations of LLMs in reasoning contexts and aids in building more interpretable and reliable LLM-powered systems.

Problem Statement

The project comprehensively analyses the reasoning capabilities of LLMs' across three axes of comparison.

- **Methodology Benchmarking**:
  - Compare the effectiveness of Retrieval-Augmented Generation with and without explicit reasoning cues
  - Contrast the overall performance of static one-shot prompting versus dynamic few-shot adaptation
  - PAL for coding in Python vs. PAL for coding in reasoning engines like Resource Description Framework (RDF)
  - Assess the effectiveness of Agentic Chain-of-Thought prompting with and without access to the internet
- **Architectural Comparisons:**
  - Compare the performance of open-source models like Llama and closed-source models such as Gemini Flash
  - Analyse the impact of parameter scaling in LLMs (e.g. 8B vs. 70B) on reasoning performance and consistency
  - Evaluate the performance of models with specialised training for conversation, coding, and reasoning
- **Generalisation Assessment:**
  - Measure the repeatability of reasoning tasks and performance drift across multiple experimental runs

Challenges in Reasoning with LLMs

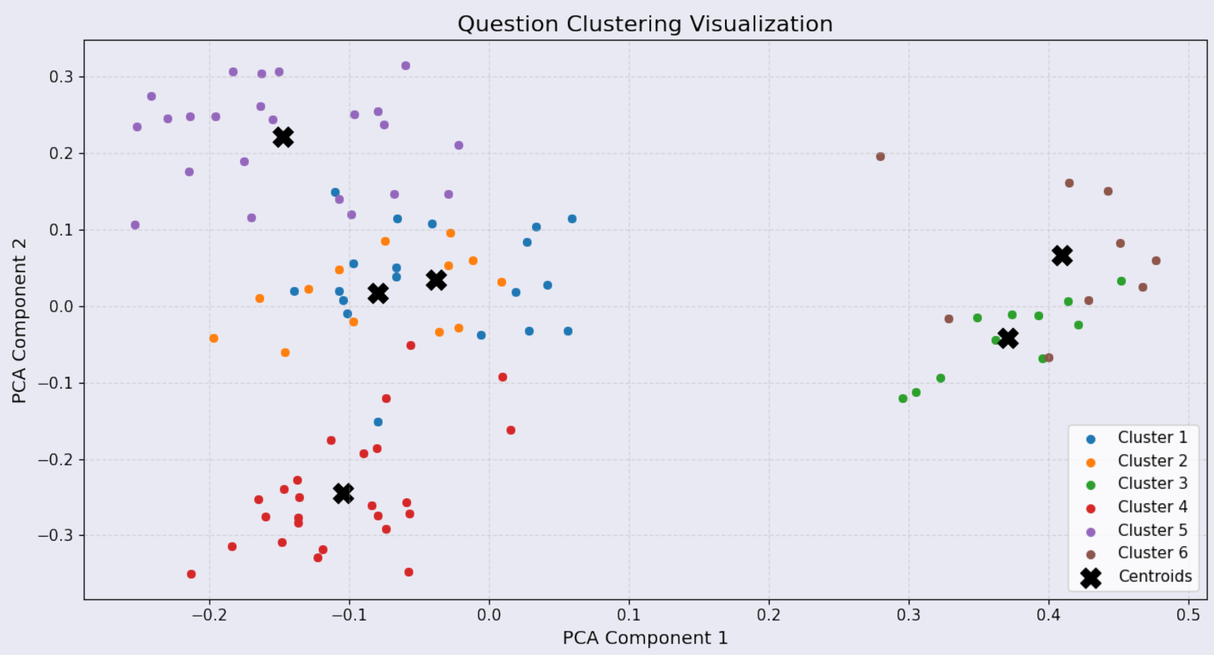
Several key challenges to achieving reliable reasoning, as highlighted in recent research, include:

- Lack of True Reasoning Abilities
- Cross-Domain Complexity
- Lack of Explainability
- Outdated Knowledge
- Mathematical Inaccuracy
- Tendency to Hallucinate
- Weaknesses in Multi-Step Problem-Solving
- Context Retention Difficulties

Methods to Aid Reasoning

- **Direct Response Generation:** LLMs attempt reasoning tasks independently, without external tools or structured guidance in this method. It is a reference to compare enhanced reasoning strategies like CoT and PAL.
  - **Baseline Response:** The baseline model is prompted without augmentation, explanations, or prompt engineering. While efficient, this approach often lacks robustness in practice and logical consistency.
  - **RAG Without Explanation:** Retrieves example problems and answers from a structured knowledge corpus without step-by-step reasoning. It may improve factual recall, but it struggles with deeper logical inference.
  - **RAG With Explanation:** Provides retrieved examples accompanied by detailed reasoning steps, enabling the model to mimic structured problem-solving processes and generalise more effectively across diverse tasks.
- **Program-Aided Language Models (PAL):** PAL methods shift reasoning to symbolic execution by prompting the language model to generate executable code (either in Python or RDF format), which is then run to produce deterministic results. This significantly reduces hallucinations and supports structured reasoning.
  - **PAL from Scratch (Python/RDF):** The model produces executable code in Python or RDF without relying on in-context examples. Python code generation tends to perform better due to the model's extensive exposure during training, whereas RDF code generation is less effective because of sparse training data.
  - **PAL with Static One-Shot Prompting for RDF Code:** In this approach, the model receives a single, fixed RDF example to guide its code generation. It performs reliably on inputs that align closely with the provided template, but struggles to adapt effectively to complex problem structures like the use of RDF syntax.
  - **Few-Shot Prompting for PAL (Python/RDF):** This technique significantly enhances reasoning by dynamically retrieving and utilising multiple relevant Python code snippets to guide reasoning. This allows the model to use contextual cues to generate and execute new code that logically solves the problem.
- **Agentic Chain-of-Thought (CoT):** This method enables language models to function as reasoning agents by breaking down complex problems into explicit, logically coherent steps. It supports both closed environments and internet-enabled configurations, thereby facilitating more structured, traceable, and effective problem-solving.
  - **Agentic CoT Without Internet:** The model operates within a self-contained environment, utilising internal tools such as calculators or memory buffers to perform reasoning in a step-by-step manner. This setup is suited for logic-based or math-focused benchmarks that do not require access to external information sources.
  - **Agentic CoT With Internet:** The model is equipped with internet access and external tools such as search engines or APIs in this setting. This extended capability enables real-time fact-checking, information retrieval, and multi-hop reasoning, making it highly effective for tackling complex, open-domain problem-solving tasks.

K-Means Clusters of Reasoning Benchmark Test Problems

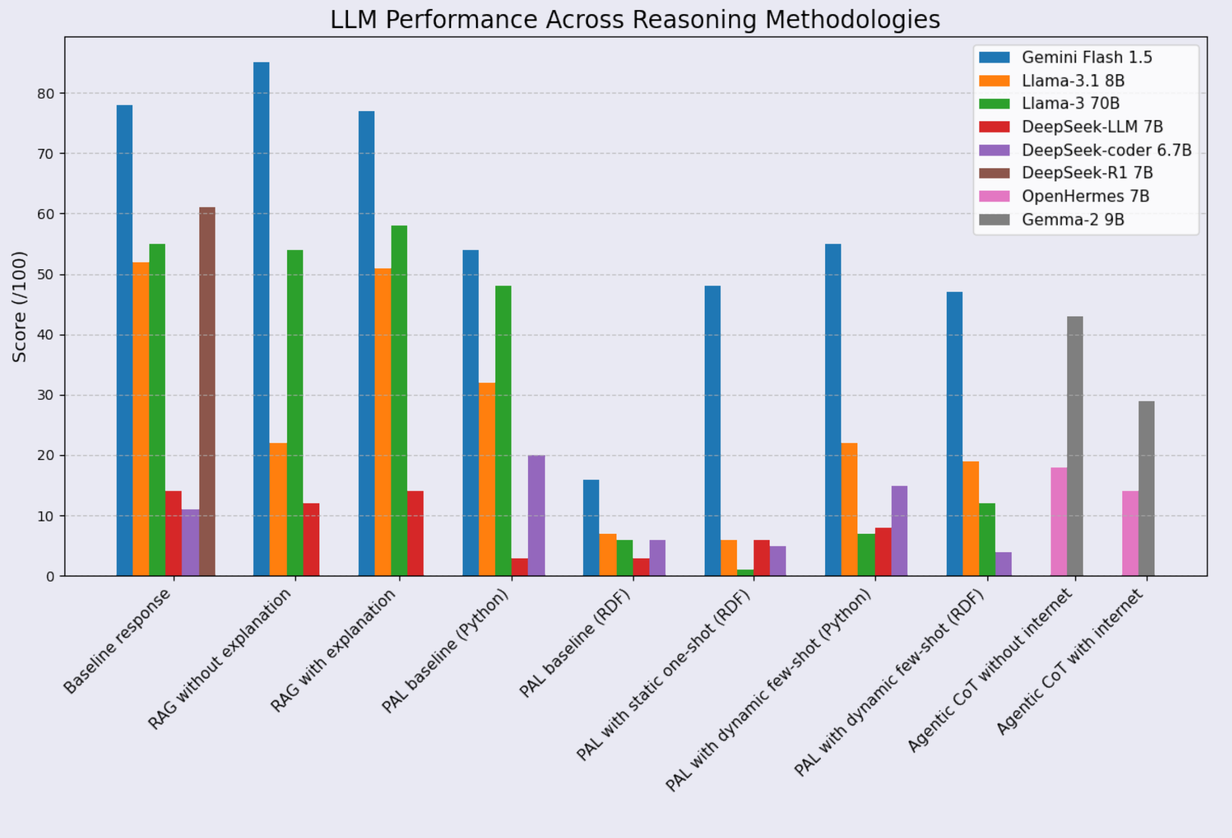


Methodology

This section details the reasoning questions, curated knowledge corpora, and evaluation techniques for assessing the language model's performance across diverse reasoning tasks.

- **Reasoning Questions:** To evaluate the language model's reasoning capabilities, we curated a dataset of 100 complex problems spanning domains like logic, probability, and constraint satisfaction. The problems were picked from established benchmarks like the American Invitational Mathematics Examination (AIME) and MATH-500, which are widely used to assess advanced reasoning models. These benchmarks represent challenging problems requiring multi-step deduction and creative problem-solving. We performed vector clustering analysis on the dataset to evaluate its diversity, confirming that it covers a broad spectrum of reasoning tasks.
- **Knowledge Corpus:** Several specialised knowledge corpora were developed to support different reasoning approaches, such as Retrieval-Augmented Generation (RAG) and Program-Aided Language Models (PAL). The RAG corpus includes two collections: one with only problems and answers, and another with problems accompanied by detailed step-by-step explanations. For PAL, we created two corpora: one with reusable Python code templates for solving reasoning problems, and another using RDF-based code for a reasoning engine.
- **Evaluation Techniques:** We implemented two automation tools to evaluate model outputs: the Regex Extractor and the PAL Executor. The Regex Extractor uses regular expressions to identify and extract final answers from model responses, ensuring accurate matching with the correct answers. The PAL Executor extracts Python code from responses, executes it in a controlled environment, and compares the results with expected outcomes. These tools ensure consistent and reproducible evaluation across different reasoning approaches.

Benchmark Scores of Different LLMs



Key Findings

The project comprehensively analyses the reasoning capabilities of LLMs' across three axes of comparison.

- **Methodology Benchmarking**:
  - **RAG with explanation vs RAG without explanation**: Closed-source models benefited more from RAG without explanations. In contrast, open-source models showed greater improvement with RAG, which included detailed reasoning. These findings underscore the need to tailor RAG strategies based on model characteristics, balancing the trade-off between example quantity and explanatory depth for optimal performance.
  - **Static One-Shot vs Dynamic Few-shot Adaptation**: The study revealed that dynamic adaptation significantly improved performance. Despite dynamic retrieval, Gemini's strong bias toward Python code showed that advanced retrieval techniques can't easily overcome model preferences built during training.
  - **PAL for Python Code Generation vs PAL for Reasoning Engine Frameworks**: While RDF frameworks hold potential for structured reasoning, current LLMs lack sufficient exposure to RDF patterns, limiting their ability to generate accurate code. This gap in exposure hinders effective code generation.
  - **Agentic CoT Prompting with and without Internet Access**: Agentic CoT performed better when operating without internet access, as the models relied solely on their internal knowledge. This setup helped avoid information overload, which led to errors and inconsistencies when the models had internet access.
- **Architectural Comparisons:**
  - **Open-source vs Closed-source Models:** Closed-source models generally outperformed open-source models, with the performance gap narrowing when advanced techniques such as RAG or CoT were applied. The gap continues to reduce with the use of larger model sizes and specialised training.
  - **Parameter Scaling Effects:** Larger models outperformed smaller ones in reasoning tasks such as Python code generation and RAG without explanation. However, both large and small models struggled with RDF code generation, emphasising the need for domain-specific training to improve performance in specialised areas.
  - **Specialised Training Effects:** Specialised training, especially reinforcement learning for reasoning, greatly enhanced performance over general-purpose or code-specialised models, though at longer inference times.
- **Generalisation Assessment:**
  - **Performance Drift Across Multiple Runs**: Despite similar average scores, run-to-run variability revealed key differences in reasoning stability, with larger models and optimised deployment platforms offering significantly higher consistency. Smaller models exhibited substantial fluctuations and prediction shifts, reinforcing the importance of scale and architecture for dependable reasoning performance.