
Teaching algorithmic reasoning via in-context Learning

Summary:

The paper introduces four key stages for successfully teaching algorithmic reasoning to LLMs:

1. Formulating Algorithmic Skills:

Using an Algorithmic Prompting (AP) Approach, i.e., a non-ambiguous description of algorithm execution based on the proposed method.

For example: When an LLM learns to add, such as $182 + 376$, it might recognize patterns that aren't the real rules. For instance, it could think, "add even numbers, carry a 1" instead of "add to over 9, carry a 1." To learn correctly, it needs to prioritize the correct rules, but current LLMs often fail to do so.

In this step, addition problems were tested on the model, focusing on adding large numbers together. Researchers experimented with different ways of presenting examples to the model. Some examples were simple, showing only a few addition problems and their answers. Others were highly detailed, demonstrating each step of the addition process, such as how to carry over numbers.

They found that when the model was provided with clear, step-by-step examples—akin to an actual math algorithm—it learned to add effectively, even with very long numbers it hadn't encountered before. However, when given vague or overly simple examples, the model performed poorly. To ensure the model wasn't merely memorizing patterns, researchers introduced examples with intentional mistakes in the steps. This experiment confirmed that the model was genuinely learning the addition algorithm from the detailed examples.

2. Teaching Multiple Skills Simultaneously (Skill Accumulation):

In this step, researchers tested whether the model could learn addition and subtraction simultaneously, including calculations involving positive and negative numbers.

They discovered that the model could indeed identify and execute the correct algorithm based on the given problem. However, subtraction proved more challenging than addition, likely due to its increased complexity (requiring more steps). Interestingly, when the model was trained on both addition and subtraction problems together, it performed better on subtraction than when trained on subtraction alone. This suggests that learning related skills reinforces understanding and improves overall performance.

3. Teaching How to Combine Skills (Skill Composition):

Researchers explored whether a model could learn complex tasks by combining simpler ones. They first taught the model basic addition, then how to add multiple numbers, and finally, how to perform multiplication through repeated addition.

They found that providing examples showing how each new skill builds upon the previous ones significantly improved learning. For instance, demonstrating how adding multiple numbers relies on basic addition helped the model grasp more complex tasks. In contrast, simply introducing multiplication without first explaining addition led to significantly poorer performance.

To address the model's limited memory (context length), researchers implemented a staged task-processing approach or used collaborative models to handle sub-tasks. This method allowed the model to process more complex problems efficiently, demonstrating that LLMs can effectively learn and combine skills, making them more capable of solving intricate tasks.

4. Teaching How to Use Skills as Tools:

Researchers tested how well a model could use a learned addition algorithm to solve complex word problems, such as those in the GSM8k dataset. These problems require both logical reasoning (figuring out the steps) and accurate calculations.

Instead of relying on external calculators, researchers focused on enhancing the model's internal arithmetic capabilities within its reasoning process. They used a filtered version of GSM8k, containing only problems solvable with addition, and created a more challenging version (GSM8k-Hard) with larger numbers.

They then evaluated the model's performance when it incorporated the learned addition algorithm into its chain-of-thought reasoning or when it was used as a tool the model could call upon. The goal was to determine whether the model could accurately perform calculations within the reasoning process required to solve these complex word problems.

Conclusion:

This paper explores how to teach AI models to perform complex tasks by providing detailed, step-by-step examples within prompts. It demonstrates that "algorithmic prompting" significantly improves performance in various reasoning tasks compared to standard methods. Furthermore, it examines how models learn multiple skills, combine them, and use them as tools, providing valuable insights into enhancing AI capabilities.

Unleashing the Potential of Prompt Engineering in Large Language Models: A Comprehensive Review

Summary:

Paper provides a thorough review of **prompt engineering**—a crucial technique for optimizing **Large Language Models (LLMs)** such as **GPT-4, Claude-3, and Vision-Language Models (VLMs)** like **CLIP and ALIGN**.

1. Fundamentals of Prompt Engineering

LLMs often generate overly generic responses if given vague instructions. Detailed, specific prompts improve output relevance (e.g., "Explain AI advancements from a hardware perspective" rather than just "Explain AI advancements").

- Clear & Precise Prompts – Avoid vague instructions; specificity improves relevance.
- Role-Prompting – Assigning roles (e.g., "You are a historian") enhances contextual accuracy.
- Triple Quotes – Helps separate prompt components for better comprehension.
- Resampling – Running prompts multiple times improves reliability.
- One-Shot & Few-Shot Prompting – Providing examples enhances performance; sometimes, zero-shot prompting works better.

2. Advanced Prompt Engineering

For complex reasoning tasks, structured prompting improves LLM performance.

- Chain-of-Thought (CoT) – Encourages step-by-step reasoning; "Let's think step by step" boosts accuracy.
 - *Zero-Shot CoT* – No examples, just reasoning cues.
 - *Golden CoT* – Uses predefined reasoning steps.
- Self-Consistency – Generates multiple reasoning paths and selects the best.
- Generated Knowledge – Model generates background info before answering.
- Least-to-Most Prompting – Breaks down complex problems into smaller steps.
- Tree/Graph of Thoughts (ToT/GoT) – Explores multiple solution paths for better decision-making.
- Decomposed Prompting (DECOMP) – Divides tasks into sub-tasks for systematic solutions.
- Active Prompting – Identifies uncertainties and adjusts prompts.
- Prompt Pattern Catalog – Collection of reusable, structured prompt templates.
- Prompt Optimization – Uses automated refinements (e.g., ProTeGi, RL).

3. Prompt Engineering in Vision-Language Models (VLMs)

Techniques tailored for multimodal AI models:

- Context Optimization (CoOp) – Learns optimal prompts automatically.
- Conditional Context Optimization (CoCoOp) – Adjusts dynamically for different inputs.
- Multimodal Prompt Learning (MaPLE) – Merges text and visual prompts for improved AI comprehension.

A Survey of Large Language Models

Summary:

LLMs have evolved significantly, demonstrating the ability to perform complex language understanding and generation tasks. The paper categorizes their development into four major stages:

1. **Statistical Language Models (SLMs)** – Used n-gram models and probability estimation for text processing.
2. **Neural Language Models (NLMs)** – Introduced deep learning approaches like word embeddings (word2vec) and Recurrent Neural Networks (RNNs).
3. **Pre-trained Language Models (PLMs)** – Used context-aware training, such as BERT and GPT-2, to enhance performance across tasks.
4. **Large Language Models (LLMs)** – Scaling PLMs led to new capabilities, such as in-context learning, seen in models like GPT-3 and GPT-4.

2. Key Advancements in LLMs

2.1 Scaling Laws

Scaling model size, data, and compute power has led to better performance.

- Kaplan Scaling Law (OpenAI): Focuses on increasing model size.
- Chinchilla Scaling Law (DeepMind): Suggests balancing model size and training data.

2.2 Emergent Abilities

LLMs display new, unexpected abilities that are not present in smaller models:

- In-context Learning (ICL) – Models like GPT-3 can perform tasks without explicit fine-tuning.

- Instruction Following – Instruction-tuned models (e.g., InstructGPT) generalize better to unseen tasks.
- Step-by-step Reasoning – Chain-of-Thought (CoT) prompting improves problem-solving accuracy.

3. Core Techniques in LLM Development

3.1 Training Strategies

- Scaling Up Models – Larger models like GPT-4 and PaLM exhibit superior performance.
- Training Data Quality – Ensuring diverse and clean datasets is critical.

3.2 Adaptation Tuning

- Instruction Tuning – Fine-tuning LLMs to follow user instructions.
- Reinforcement Learning with Human Feedback (RLHF) – Used in ChatGPT and InstructGPT for better user alignment.

3.3 LLM Alignment & Safety

- Mitigating Bias & Toxicity – RLHF and adversarial training improve safety.
- Fine-tuning for Ethical AI – Ongoing research on human-AI alignment.

4. Applications of LLMs

LLMs are revolutionizing multiple domains:

- NLP Tasks – Machine translation, text summarization, and content generation.
- Code Generation – Models like Codex and StarCoder excel at programming tasks.
- Multimodal AI – Vision-language models (GPT-4V) integrate text and images.
- AI Assistants – Used in chatbots (ChatGPT), search engines (New Bing), and office automation (Microsoft Copilot).

5. Future Directions & Challenges

- Enhancing Model Efficiency – Reducing compute costs while improving performance.
- Improving Interpretability – Understanding how LLMs develop emergent abilities.
- Addressing Data Scarcity – Finding new methods for data augmentation.
- Exploring AGI Potential – LLMs are pushing the boundaries of Artificial General Intelligence (AGI).

Conclusion

This survey presents an in-depth review of LLMs, highlighting scaling effects, emergent behaviors, and adaptation strategies. Future research must focus on ethical AI development, improved efficiency, and safety alignment to maximize LLM potential.

GPT-3.5, GPT-4, or BARD? Evaluating LLMs Reasoning Ability in Zero-Shot Setting and Performance Boosting Through Prompts

Summary:

This paper evaluates the reasoning abilities of ChatGPT-3.5, ChatGPT-4, and Google's BARD in a zero-shot setting across multiple reasoning domains. It also proposes engineered prompts to enhance model performance.

Reasoning Performance Comparison

- GPT-4 outperforms GPT-3.5 and BARD across most reasoning tasks.
- BARD performs better in some cases, particularly in multi-hop reasoning without prompt engineering.
- All three models struggle with inductive, mathematical, and multi-hop reasoning tasks.

Evaluation Across Reasoning Domains

- Deductive Reasoning: GPT-4 achieved higher accuracy than GPT-3.5 and BARD.
- Inductive Reasoning: All models struggled, but prompt engineering improved GPT-4's performance significantly.
- Abductive Reasoning: GPT-4 performed best, but all models made similar errors in nuanced reasoning.
- Mathematical Reasoning: GPT-4 was better than GPT-3.5 and BARD, but struggled with complex calculations.
- Commonsense Reasoning: GPT-4 performed best, but all models failed in some basic commonsense tasks.
- Causal Reasoning: GPT-4 had slightly better accuracy, but all models occasionally misidentified causal relationships.
- Multi-hop Reasoning: BARD outperformed GPT models, but all models struggled with long-context reasoning.

Effectiveness of Prompt Engineering

- Properly crafted prompts significantly improve performance in zero-shot settings.

- GPT-4 benefits the most from prompt engineering, achieving near-perfect scores in some tasks.
- BARD performs well without strong prompt engineering but still lags behind GPT-4.

Conclusion & Future Directions

- GPT-4 is the most capable model overall, but all LLMs struggle with complex reasoning tasks.
- Inductive and mathematical reasoning require significant improvement in all models.
- Prompt engineering is an effective way to boost performance, especially for zero-shot reasoning.
- Future research should explore better reasoning mechanisms and expand benchmarks for more diverse tasks.

Teaching Arithmetic to Small Transformers

Summary:

The research highlights that conventional training data is not optimal for arithmetic learning, and that simple formatting changes can significantly improve accuracy. The authors demonstrate that training on chain-of-thought style data, which includes intermediate step results, enhances accuracy, sample complexity, and convergence speed, even without pretraining.

Key Findings:

- 1. Training Data Format Matters**
 - Conventional data formats are inefficient for teaching arithmetic to transformers.
 - Simple formatting modifications, such as reversing the output order (least significant digit first), significantly improve accuracy and learning efficiency.
- 2. Emergence of Arithmetic Abilities**
 - Learning addition exhibits a sharp phase transition when the training data reaches a sufficient size.
 - This phenomenon is linked to low-rank matrix completion, suggesting that arithmetic learning in transformers follows principles of structured data representation.
- 3. Chain-of-Thought (CoT) Training Improves Learning**

- Providing intermediate steps in a structured "scratchpad" format enhances accuracy, sample efficiency, and convergence speed.
 - Detailed step-by-step instructions reduce the number of training examples needed for perfect arithmetic performance.
4. **Influence of Model Scale and Pretraining**
- Small transformers (NanoGPT, GPT-2) can learn arithmetic operations even without pretraining.
 - Fine-tuning pretrained models can accelerate learning, but inconsistent formatting between pretraining and fine-tuning may reduce performance.
5. **Generalization and Robustness**
- Models trained on arithmetic data alone can generalize to unseen numbers, outperforming traditional low-rank matrix completion techniques.
 - Introducing noise into training data (e.g., randomizing intermediate steps) slows learning but does not prevent models from eventually reaching full accuracy.

Implications:

- The research highlights the importance of well-structured, instructive training data over sheer model size.
- Insights from this study can improve arithmetic reasoning in small language models, making them more efficient for specific applications requiring numerical computations.

Exposing Attention Glitches with Flip-Flop Language Modeling Summary:

Summary:

This paper investigates why large language models (LLMs) sometimes produce factual inaccuracies, particularly in long reasoning chains. It introduces the concept of **attention glitches**, where Transformer-based models fail to robustly process long-range dependencies due to the architecture's inductive biases.

Key Contributions:

1. **Identification of Attention Glitches**
 - Transformers can exhibit sporadic and unpredictable reasoning errors, even in simple algorithmic tasks.
 - These glitches may contribute to "closed-domain hallucinations" in LLMs, where models generate factually incorrect responses despite their coherence.

2. Flip-Flop Language Modeling (FFLM) Benchmark

- A synthetic benchmark designed to test a model's ability to store and retrieve a single bit across long sequences, a fundamental reasoning task.
- Transformers struggle with this task, making sporadic errors, while recurrent models like LSTMs perform perfectly.

3. Empirical Findings

- Transformers exhibit a long tail of errors when processing long sequences, even when trained on large datasets.
- LSTMs outperform Transformers in tasks requiring memory retention, suggesting that recurrence may be essential for robust reasoning.
- Scaling and regularization techniques help but do not eliminate errors, even with larger models and better training methods.

4. Potential Mitigations

- Data augmentation: Training on a more diverse set of sequences significantly reduces errors.
- Attention sharpening: Regularization techniques that enforce sparsity in attention weights improve performance but still do not eliminate glitches.
- Architectural improvements: The authors suggest that incorporating recurrence into Transformers (or hybrid models) might be necessary for fully solving these issues.

Implications:

- The study suggests that attention glitches could be a key reason for hallucinations in natural LLMs.
- Future work should explore hybrid architectures combining Transformers and recurrent models to enhance reliability.
- Flip-flop benchmarks provide a new way to stress-test reasoning capabilities in LLMs.

Can Neural Networks Do Arithmetic?

Summary:

The paper examines the challenges neural network models, particularly large language models (LLMs), face in numerical reasoning and arithmetic operations. While LLMs have demonstrated impressive mathematical reasoning skills, they still struggle with accurately manipulating large numbers and performing arithmetic operations, particularly when problems are posed differently than those seen in training.

Key findings include:

- **Limitations in Number Representation:** Neural networks improve in handling small and medium-sized numbers, but their performance declines for large quantities not seen in training.
- **Arithmetic Performance:** Despite advances, even state-of-the-art models like Minerva and GPT-4 fail at higher-digit arithmetic problems. For example, Minerva achieves over 80% accuracy for 10-digit addition but drops significantly for larger numbers.
- **Extrapolation Challenges:** LLMs struggle with generalizing arithmetic operations to numbers beyond their training distribution. External calculators and algorithmic approaches are necessary to improve performance.
- **Benchmarks and Performance:** Models show steady improvement on benchmarks like GSM8K, but their understanding of numerical reasoning remains limited, particularly in cases requiring compositionality and magnitude estimation.

The study highlights that while deep learning models excel in many reasoning tasks, mastering numerical understanding and arithmetic remains an open challenge

Algorithm of Thoughts – Enhancing Exploration of Ideas in Large Language Models

Summary:

The paper proposes the *Algorithm of Thoughts (AoT)*, a novel prompting strategy designed to improve reasoning capabilities in Large Language Models (LLMs). Unlike existing methods such as *Chain of Thought (CoT)* and *Tree of Thought (ToT)*, AoT integrates algorithmic structures into in-context learning to enhance idea exploration and problem-solving efficiency.

Key Contributions

1. **Efficiency Over Traditional Methods**
 - AoT allows LLMs to navigate complex reasoning tasks with fewer queries than multi-query methods like ToT.
 - It outperforms CoT and ToT while using significantly fewer tokens and computational resources.
2. **Algorithmic Structuring in Reasoning**
 - AoT follows an exploratory reasoning process similar to tree-search algorithms (e.g., DFS, BFS) but keeps everything in a single query.

- Unlike CoT, which follows a strict step-by-step progression, AoT dynamically explores multiple solution pathways within a single generation.

3. Performance in Problem-Solving Tasks

- The paper tests AoT in various tasks, including:
 - Game of 24 (Mathematical Reasoning): AoT achieves 71% success with a single query, outperforming ToT.
 - Mini Crosswords (Linguistic Reasoning): AoT achieves a 52% word success rate, surpassing previous methods.
 - Question Answering (GSM8K & StrategyQA): AoT provides a slight boost over CoT and ToT in factual reasoning.
 - Dynamic Programming Problems (Coin Change & Edit Distance): AoT significantly outperforms CoT in structured problem-solving.

4. Reduction in Computational Costs

- AoT reduces the number of LLM queries by over 100x compared to ToT while maintaining higher accuracy.
- This leads to lower financial costs, reduced latency, and decreased energy consumption.

Key Findings

- LLMs can exceed the efficiency of the algorithms they are trained with by leveraging their generative recurrence.
- AoT enables a balance between structured exploration and model intuition, making it more adaptable than rigid tree-search methods.
- Fine-tuning does not replace AoT's advantages, highlighting the importance of structured in-context learning.

Parsel – Algorithmic Reasoning with Language Models by Composing Decompositions

Summary:

The paper introduces Parsel, a framework that enhances Large Language Models (LLMs) in solving hierarchical multi-step reasoning tasks, particularly for algorithmic problem-solving and code generation. Instead of directly generating entire programs, Parsel decomposes problems into modular components, implements these components separately, and then combines them into a complete solution.

How It Works:

1. **Decomposition:** The LLM splits a problem into structured function descriptions in natural language.
2. **Implementation:** Each function is generated separately as a modular Python function.
3. **Composition and Validation:** The Parsel Synthesizer tests different function implementations and combines them into a correct program.

Key Benefits:

- **Higher accuracy:** Parsel achieves a 75 percent higher pass rate on competition-level APPS problems compared to Codex and AlphaCode.
- **Better efficiency:** Instead of regenerating full programs, Parsel optimizes function reuse.
- **Improved reasoning:** By structuring tasks hierarchically, Parsel helps LLMs handle long, complex problems more effectively.
- **Stronger results:** Improves HumanEval pass@1 from 67 percent to 85 percent, outperforming GPT-4 alone.

Language Models Are Weak Learners

Summary:

This paper proposes a looped transformer architecture designed to improve in-context learning by integrating iterative computation. Standard transformers lack an inherent iterative structure, making them inefficient at emulating traditional machine learning algorithms that rely on iterative optimization. The authors demonstrate that looped transformers can effectively learn iterative algorithms while requiring significantly fewer parameters than standard transformers.

Key Contributions

1. **Looped Transformer Architecture**
 - Unlike standard transformers, looped transformers repeatedly apply the same computation across multiple iterations, mimicking the behavior of iterative algorithms.
 - This design enables better parameter efficiency and the ability to solve complex tasks with fewer layers.
2. **Training Methodology**
 - The authors introduce a new training strategy for looped transformers to ensure stable and efficient convergence.

- As the number of loop iterations increases, the model progressively refines its predictions, allowing it to approximate fixed-point solutions.
- 3. **Performance on Learning Tasks**
 - The looped transformer matches or outperforms standard transformers in in-context learning for linear regression, decision trees, and neural networks.
 - Despite using only 10% of the parameters, it achieves comparable accuracy to a 12-layer transformer.
 - It generalizes well across different function types, including sparse linear functions and decision trees.
- 4. **Parameter Efficiency & Model Complexity**
 - The looped transformer achieves a 12x reduction in parameter count while maintaining performance on data-fitting tasks.
 - It effectively emulates fixed-point iteration methods, enabling it to learn optimization algorithms efficiently.

Implications

- Looped transformers can replace deeper standard transformers in scenarios where iterative learning is needed, improving efficiency.
- The architecture offers a promising direction for reducing computational costs while maintaining or improving learning capabilities.
- Future research can extend looped transformers to more complex reasoning and optimization problems.

Complex QA & Language Models Hybrid Architectures – A Survey

Summary:

This paper provides a comprehensive survey on complex question-answering (CQA) systems and how hybrid architectures combining Large Language Models (LLMs) with external components can improve accuracy, reasoning, and domain adaptation. While LLMs like GPT-4 and BLOOM perform well on standard QA tasks, they struggle with multi-step reasoning, fact verification, and long-form answers. The survey reviews existing limitations, evaluation techniques, and potential hybridization strategies to enhance complex QA performance.

Key Contributions

1. **Challenges in Complex QA**
 - Standard LLMs struggle with multi-step reasoning, long-form QA, and domain-specific adaptation.

- Issues include hallucinations, explainability, data sensitivity, and truthfulness.
- 2. **Evaluation of LLM Capabilities**
 - Benchmarks from HELM, BLOOM, and BIG-bench highlight the strengths and weaknesses of LLMs.
 - Key evaluation metrics include accuracy, calibration, robustness, fairness, and efficiency.
- 3. **Hybrid Architectures for Complex QA**
 - Integrating LLMs with external tools (e.g., knowledge graphs, symbolic AI, multimodal search) improves reasoning and factual accuracy.
 - Prompting strategies, reinforcement learning (RLHF), and retrieval-augmented generation (RAG) can enhance QA performance.
- 4. **Training and Adaptation Techniques**
 - Methods like fine-tuning, instruction tuning, and domain adaptation help tailor LLMs for specific industries (e.g., medicine, finance).
 - Combining neuro-symbolic approaches and program synthesis can improve logical reasoning.
- 5. **Future Research Directions**
 - Addressing bias, hallucinations, and multi-modality remains a key challenge.
 - More robust human-in-the-loop frameworks are needed for better control and verification of AI-generated answers.

Implications

- Hybrid architectures combining LLMs with structured knowledge bases are more effective for complex QA.
- Multi-step reasoning and decomposition techniques can significantly improve accuracy in long-form QA.
- The survey emphasizes that LLMs alone are not sufficient for complex problem-solving, and integrating external reasoning modules is crucial.

Answering Causal Questions with Augmented LLMs

Summary:

This paper investigates how **Large Language Models (LLMs) can be augmented** to improve their ability to answer **causal questions**, which they inherently struggle with. The authors explore two augmentation methods:

1. **Context Augmentation** – Providing causal graphs and treatment effect data directly in the LLM's prompt.

2. **Tool Augmentation** – Giving LLMs access to an external API that processes causal queries.

The study evaluates how these approaches enable LLMs to answer **causal reasoning questions** and highlights their limitations.

Key Contributions

- **Comparison of Augmentation Methods:** The study contrasts context-based augmentation (embedding causal knowledge in prompts) with tool-based augmentation (calling external APIs).
- **Evaluation of Causal Question Answering:** The paper tests these methods on synthetic causal graphs and treatment effect datasets.
- **Findings on LLM Capabilities:** It demonstrates that LLMs alone cannot reliably perform causal reasoning and need access to external tools for better accuracy.

Experiments and Findings

1. **Context-Augmented LLMs**
 - Directly embedding causal graphs and treatment effects in the prompt.
 - Struggles with longer context lengths, leading to decreased performance as the problem size increases.
 - Makes more reasoning errors and is less reliable for complex causal inference.
2. **Tool-Augmented LLMs (API Access)**
 - Uses an external tool to process causal graphs and treatment effects.
 - More consistent performance, as it offloads complex reasoning tasks to a structured algorithm.
 - More robust to increasing problem complexity compared to context augmentation.
3. **Quantitative Evaluation**
 - The tool-based method significantly outperformed context augmentation, especially for larger causal graphs.
 - Both methods showed JSON formatting errors, but tool augmentation had a higher correctness rate.

Conclusions

- LLMs alone are insufficient for answering causal questions reliably.
- Tool augmentation is a more effective way to enhance LLMs for causal reasoning.
- Future work should focus on integrating expert causal inference systems into LLM workflows for more robust decision-making.

Reasoning in Large Language Models – A Geometric Perspective

Summary:

This paper explores the reasoning capabilities of Large Language Models (LLMs) from a geometric perspective, focusing on how self-attention mechanisms shape the expressive power of LLMs. The authors establish a relationship between the density of attention graphs and the intrinsic dimension of the model's internal representations, arguing that higher intrinsic dimensions correlate with better reasoning abilities.

Key Contributions

1. Geometric View of LLM Reasoning

- The authors analyze the intrinsic dimension of attention layers, showing that denser self-attention graphs lead to richer representations in the multi-layer perceptron (MLP) layers.
- A higher intrinsic dimension means that the model can better approximate complex reasoning tasks.

2. Impact of Context Length and Attention Heads

- Increasing context length and number of attention heads enhances the intrinsic dimension of LLMs, leading to better approximation and reasoning.
- The study connects this observation to few-shot prompting, explaining why longer prompts improve reasoning performance.

3. Empirical Findings on LLM Reasoning

- The authors conduct experiments using the GSM8K-Zero dataset to evaluate the impact of intrinsic dimension on reasoning accuracy.
- Results show that models with higher intrinsic dimension in their final layers answer questions more accurately.
- Random token experiments confirm that simply increasing input size does not always improve reasoning—it must be meaningful information.

Conclusions

- LLM reasoning ability is strongly linked to its internal geometric structure.
- Denser self-attention graphs and higher intrinsic dimensions improve logical reasoning and approximation capabilities.
- Designing LLM architectures with a focus on increasing intrinsic dimension (e.g., through more attention heads or better training strategies) could enhance reasoning without just increasing model size.

A Human-Like Artificial Intelligence for Mathematics

Summary:

This paper explores the idea of designing **human-like AI systems** for mathematics, arguing that artificial intelligence should not be purely **axiomatic and deductive** but should incorporate aspects of **human mathematical cognition**. The author presents six key reasons why AI should adopt human-like reasoning rather than relying solely on brute-force computation or formal logic.

Key Contributions

1. **Mathematical Cognition and Human-Like AI**
 - Human cognition, despite its limitations and biases, has led to the creation of insightful mathematical discoveries.
 - Mathematics is not just deductive proofs but also involves creativity, intuition, and problem structuring.
2. **The Role of Human Cognitive Features in Mathematics**
 - The paper discusses various cognitive traits essential for mathematical thinking, such as pattern recognition, spatial reasoning, working memory, and metacognition.
 - Human cognition can detect structure in real-world problems and efficiently approximate solutions.
3. **Comparison Between AI and Human Mathematicians**
 - Current AI systems, such as LLMs and symbolic solvers, can perform calculations but lack deep mathematical intuition.
 - AI's reliance on pattern matching and brute-force search makes it fundamentally different from human mathematicians, who often rely on heuristics, analogy, and creative leaps.
4. **Six Arguments for a Human-Like AI Mathematician**
 - Human cognition creates mathematics: AI should reflect the cognitive processes that led to mathematical discoveries.
 - Mathematics is insightful, not just deductive: AI should develop an ability to generate new insights.
 - Human cognition detects patterns in the real world: AI should be able to formulate new problems, not just solve predefined ones.
 - Handling complexity and intuition: Human-like AI would be better at recognizing when a problem is complex and where to focus computational resources.
 - Creativity and curiosity: AI mathematicians should have an ability to explore new ideas beyond existing formulas.

- Ethical considerations: AI in mathematics should account for ethical issues, such as fairness in AI-driven mathematical applications.

Integrating Learning and Reasoning for Large Language Models and Recommender Systems

Summary:

This dissertation explores the integration of learning and reasoning in Large Language Models (LLMs) and recommender systems. While modern recommendation models and LLMs have advanced significantly, they often lack robust reasoning capabilities, which limits their ability to provide accurate and explainable recommendations. This research focuses on enhancing reasoning abilities in these models through counterfactual reasoning, logical reasoning, and generative recommendation techniques.

Key Contributions

- 1. Counterfactual Collaborative Reasoning (CCR) for Recommender Systems**
 - Introduces Counterfactual Collaborative Reasoning (CCR), a method that enhances recommendation performance by generating counterfactual training data.
 - Unlike traditional recommendation models that rely on implicit feedback (clicks, purchases), CCR leverages explicit feedback (likes, dislikes) to generate more informative counterfactual examples.
 - Experimental results on real-world datasets show that CCR improves both accuracy and transparency of recommendations.
- 2. Logical Large Language Models (L3M) for Symbolic Reasoning**
 - Proposes the Logical Large Language Model (L3M), which enhances LLMs with logical reasoning abilities.
 - L3M is trained using logical prompts to improve its ability to solve symbolic reasoning tasks, arithmetic problems, and question-answering challenges.
 - Evaluation shows that L3M significantly improves symbolic reasoning compared to standard LLMs.
- 3. Generative Recommendation (GenRec) Using LLMs**
 - Develops GenRec, a novel generative recommendation approach that uses pure text-based item representations instead of manually designed user-item IDs.
 - By leveraging LLMs, GenRec provides personalized and context-aware recommendations with enhanced flexibility.
 - This method eliminates the need for complex indexing techniques while improving user experience.

4. **MoralBench: Ethical Evaluation of LLMs**

- Introduces MoralBench, a benchmark designed to assess the ethical reasoning capabilities of LLMs.
- Evaluates how well LLMs align with moral values, fairness, and ethical norms.
- The benchmark helps identify potential biases in LLM outputs and ensures responsible AI development.

Conclusion & Implications

- Integrating reasoning mechanisms (counterfactual, logical, generative) improves the performance, transparency, and adaptability of recommender systems and LLMs.
- Future AI models should combine structured reasoning with deep learning to enhance their ability to explain and justify their decisions.
- The dissertation highlights ethics in AI, emphasizing that performance optimization must be balanced with fairness and accountability.

Diversity-Seeking Chain-of-Thought Reasoning in Large Language Models

Summary:

This thesis investigates how Generative Flow Networks (GFlowNets) can enhance chain-of-thought (CoT) reasoning in Large Language Models (LLMs). Traditional reinforcement learning approaches to fine-tuning LLMs often suffer from mode collapse, where the model generates a small number of highly similar responses. GFlowNets are introduced as a diversity-seeking alternative, allowing LLMs to generate multiple valid reasoning paths for complex problems, improving both correctness and explanation diversity.

Key Contributions

1. **Application of GFlowNets to LLM Reasoning**
 - Introduces a method for fine-tuning LLMs as GFlowNets to enhance their ability to generate diverse reasoning chains.
 - Addresses mode collapse in reinforcement learning-based LLM fine-tuning.
2. **Empirical Evaluation on GSM8K (Math Word Problems)**
 - Demonstrates that GFlowNet-fine-tuned (GFN-FT) LLMs achieve greater diversity in reasoning paths while maintaining or improving correctness.
 - Temperature variation experiments show that models fine-tuned with GFlowNets generate more varied rationales than baseline models.

3. **Enhancing In-Context Learning and Chain-of-Thought Prompting**
 - Studies how prompting techniques such as in-context learning (ICL) and chain-of-thought reasoning interact with GFlowNet fine-tuning.
 - Suggests that generating multiple reasoning paths can improve LLM robustness and adaptability to different problems.
4. **Limited-Data Fine-Tuning for Efficient Training**
 - Demonstrates that GFlowNet fine-tuning can improve LLM reasoning with very few training examples (only 16 examples used for training).
 - Suggests that fine-tuning small models on limited data can still lead to large performance gains.

Results & Findings

- **Correctness Analysis:**
 - Fine-tuned models outperform baseline models at lower temperatures (i.e., when the model is forced to generate more deterministic responses).
 - After 10 epochs of fine-tuning, the model achieves the highest accuracy.
 - Longer fine-tuning (23+ epochs) decreases correctness, likely due to instability in reinforcement learning.
- **Diversity of Reasoning Paths:**
 - Models trained with GFlowNets generate more varied explanations, especially at higher temperatures.
 - Diversity remains consistently higher than standard LLMs across all temperature settings.
- **Tradeoff Between Diversity and Correctness:**
 - More diverse models sometimes generate incorrect reasoning paths, but the overall improvement in explanation diversity is beneficial for applications requiring multiple perspectives.

Example: Game of 24 (Mathematical Reasoning)

In the *Game of 24*, given four numbers, the task is to use basic arithmetic operations (+, −, ×, ÷) to make the result **24**.

◆ Standard Prompting:

Input: 8, 6, 4, 4

Output: $(4 + (8 - 6)) \times 4 = 24$

◆ Chain of Thought (CoT):

Breaks the problem into step-by-step reasoning.

Steps:




1. $8 - 6 = 2$
2. $4 + 2 = 6$
3. $6 \times 4 = 24$

Final Answer: $(4 + (8 - 6)) \times 4 = 24$

◆ Algorithm of Thoughts (AoT):

AoT **explores multiple paths**, evaluates their potential, and refines the solution dynamically.

1. Explores different arithmetic combinations

- $(8 - 6) = 2 \rightarrow (4 + 2) = 6 \rightarrow (6 \times 4) = 24$ 
- $(4 \div 4) = 1 \rightarrow (6 \times 4) = 24$ 
- $(8 \times 3) = 24$  (not valid)

2. Backtracks & selects the best path dynamically

3. Final Answer: $(4 + (8 - 6)) \times 4 = 24$

AoT achieves a **higher success rate than CoT and ToT** while using significantly fewer queries.