EXAGEN : Expert Augmented Generation System

Interdisciplinary Dual Degree Phase I

Submitted by:

Dhruvkumar Patil

200100056

Under the guidance of:

Prof. Asim Tewari

Center for Machine Intelligence and Data Science

Indian Institute of Technology Bombay

2024

Abstract

The goal of this thesis is to develop an expert system for solving mathematical problems across various domains, focusing on classifying problem complexity and ambiguity to enhance the accuracy of AI-driven solutions. Traditional AI and NLP models struggle with mathematical tasks due to the need for precise, multi-step reasoning and handling ambiguous language. This work leverages large language models (LLMs) alongside symbolic computation tools like SymPy to address these limitations, combining language understanding with exact symbolic calculations. Through multi-criteria classification, the system categorizes problems by type, complexity, and required operations, creating tailored solving pathways for each. This research is intended to support adaptive learning environments, such as Moodle, by generating personalized assessments based on difficulty, and to improve AI-based problem-solving systems for both educational and practical applications.

Keywords: Mathematical Problem Solving, Expert System, Large Language Models (LLMs), Symbolic Computation, Problem Classification

Contents

1. Introduction

In recent years, Natural Language Processing has seen significant progress with models like Transformers, BERT, and GPT. These models have changed how computers understand and work with human language, leading to advancements in things like language translation and text summarization. However, despite all these improvements, solving math problems with NLP is still quite challenging. The main reason for this is that math isn't just about understanding language—it requires working with numbers and symbols too, which requires precision that current NLP models struggle to achieve.

Math word problems are particularly difficult because they combine language with mathematical expressions. To solve these, the system needs to understand both the text and the math behind it. While models like GPT and BERT are good at recognizing patterns in text, they often fail when it comes to solving math problems that need step-by-step reasoning and exact calculations.

On the other hand, tools like Wolfram Alpha and SymPy are great at handling exact symbolic computations. They can solve algebraic equations, perform differentiation, and work with integrals with high accuracy. But, these tools need the problem to be written in a formal mathematical format. They are not able to process natural language descriptions of problems, especially if the wording is unclear or ambiguous.

This gap between language understanding in NLP and symbolic solving in tools like SymPy has led to attempts at combining the two. Some systems use large language models (LLMs) to interpret math word problems and then use symbolic computation tools to solve them. While these systems are promising, they still struggle with more complex problems, especially those that involve multiple steps, ambiguous wording, or complex mathematical functions.

The goal of this thesis is to develop an expert system that can address these challenges by combining multi-criteria classification with symbolic computation. The system will classify math problems based on different factors like ambiguity, complexity, and the number of operations or functions involved. By combining the strengths of LLMs in understanding language with the precision of symbolic tools like SymPy, the system will be able to handle a wide variety of math problems, from simple equations to more advanced calculus.

This system will also be useful in educational platforms like Moodle. It can generate custom test questions based on the classification of the problem and the skill level of the student. This will help teachers create better tests and allow students to learn at their own pace with problems suited to their abilities.

In summary, this thesis aims to bridge the gap between NLP and symbolic computation by creating an expert system that can interpret math word problems and solve them accurately. By combining LLMs with symbolic math tools, this system will offer a better solution for solving math problems, whether in education or real-world applications.

2. Literature Review

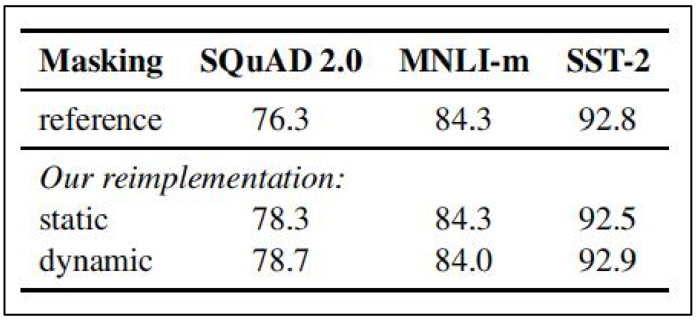
2.1 Natural Language Processing and Symbolic Mathematics

2.1.1 ROBERTa: A Robustly Optimised BERT Pretraining Approach

In the paper, the author has used dynamic masking in place of static masking used in BERT. The two types of masking can be defined as follows:

1. Static Masking: With static masking, a fixed set of tokens in each sentence is masked right at the beginning of training. This set remains the same throughout, meaning the same words are masked every time the model sees a given sentence. This approach can restrict what the model learns, as it doesn't get the chance to see varied masked tokens across training rounds.
2. Dynamic Masking: RoBERTa introduced dynamic masking to make this process more flexible. Here, tokens are randomly selected for masking on-the-fly each time the model processes a sentence. In other words, the masked words are different each time the model encounters the same sentence. This variety helps RoBERTa learn to predict masked words in a wider range of contexts, which makes it better at generalizing across language patterns.

By training on much larger datasets and using dynamic masking, RoBERTa gains from seeing more diverse masked positions than BERT, which contributes to better performance on NLP benchmarks.

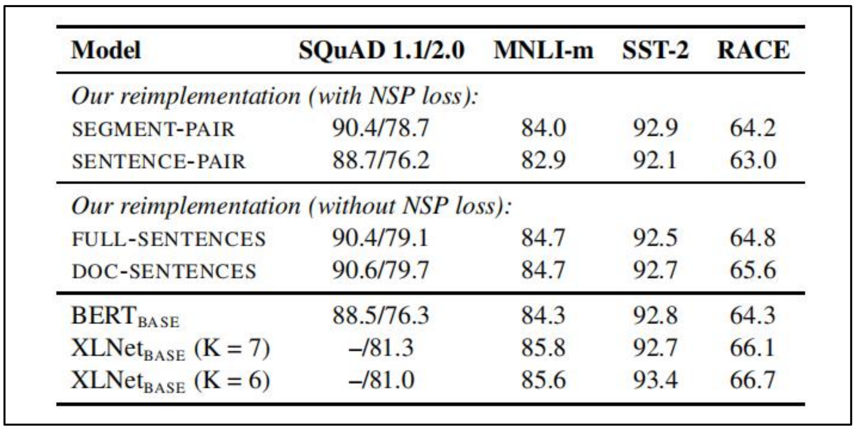


Masking	SQuAD 2.0	MNLI-m	SST-2
reference	76.3	84.3	92.8
Our reimplementation:			
static	78.3	84.3	92.5
dynamic	78.7	84.0	92.9

Table 2.1. Comparison between static and dynamic masking for BERTBASE . This report F1 for SQuAD and accuracy for MNLI-m and SST-2. Reported results are medians over 5 random initializations (seeds). Reference results are from Yang et al. (2019)

Unlike BERT, RoBERTa skips the Next Sentence Prediction (NSP) task. BERT combines NSP with MLM (Masked Language Model) to teach the model sentence relationships, but RoBERTa's creators found NSP added little to the model's effectiveness, sometimes even reducing it. So, RoBERTa sticks only to MLM, using dynamic masking and training on full sentences from huge datasets to learn broader language contexts without extra NSP steps.

This simplified approach of dynamic masking and focusing solely on MLM, helped RoBERTa outperform BERT in many tasks like text translation and classification, showing that with enough data variety, the MLM objective alone can be very powerful for building robust language understanding.



Model	SQuAD 1.1/2.0	MNLI-m	SST-2	RACE
Our reimplementation (with NSP loss):				
SEGMENT-PAIR	90.4/78.7	84.0	92.9	64.2
SENTENCE-PAIR	88.7/76.2	82.9	92.1	63.0
Our reimplementation (without NSP loss):				
FULL-SENTENCES	90.4/79.1	84.7	92.5	64.8
DOC-SENTENCES	90.6/79.7	84.7	92.7	65.6
$\text{BERT}_{\text{BASE}}$	88.5/76.3	84.3	92.8	64.3
$\text{XLNet}_{\text{BASE}}$ (K = 7)	–/81.3	85.8	92.7	66.1
$\text{XLNet}_{\text{BASE}}$ (K = 6)	–/81.0	85.6	93.4	66.7

Table 2.2. Development set results for base models pretrained over BOOKCORPUS and WIKIPEDIA. All models are trained for 1M steps with a batch size of 256 sequences. These report F1 for SQuAD and accuracy for MNLI-m, SST-2 and RACE. Reported results are medians over 5 random initializations (seeds). Results for BERTBASE and XLNetBASE are from Yang et al. (2019).

Note: RoBERTa does not focus on symbolic mathematics directly but sets the foundation for pretraining models to handle complex text. It's architecture could be adapted to math problems by learning from text-to-math conversions.

2.1.2 Deep Learning for Symbolic Mathematics

Lample and Charton present a significant advancement in the field of deep learning applied to symbolic mathematics. Their research introduces a neural network approach that utilizes a tree structure to represent complex mathematical expressions, enabling the systematic handling of operations like integration and solving ordinary differential equations (ODEs). Each node in their tree corresponds to an operator or function, while branches represent operands or sub-expressions. This hierarchical structure not only facilitates the management of nested functions but also enhances the model's ability to interpret and simplify complex mathematical expressions, similar to existing symbolic computation tools like SymPy.

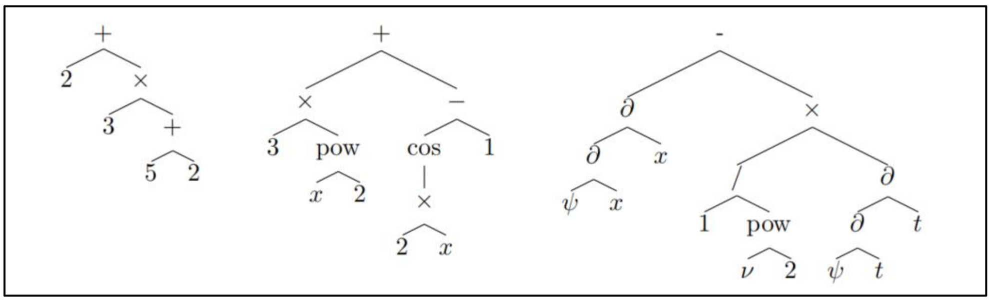


Figure 2.1. Tree structure for mathematic expressions proposed by Lample and Charton

A key aspect of their methodology is the generation of training datasets through recursive combinations of functions and constants, which allows for the creation of thousands of symbolic expressions paired with their correct solutions. This focus on symbolic tasks requiring exact solutions addresses a critical need for precision in mathematical operations, which traditional AI models often struggle to achieve due to their reliance on approximations.

Lample and Charton's findings demonstrate that deep learning models can achieve high accuracy on symbolic tasks, showcasing their potential for handling structured mathematical reasoning. Their work emphasizes the importance of precise symbolic representation in enabling neural networks to perform complex calculations effectively.

Despite the strengths of Lample and Charton's approach, there are notable gaps that can be addressed:

1. Generalization Limitations: Their model heavily relies on exact function mappings, which may hinder its ability to generalize across diverse types of mathematical problems. This limitation can be particularly challenging when dealing with problems that involve ambiguity or require creative problem-solving approaches.
2. Scope of Mathematical Tasks: The focus of their study is primarily on specific symbolic tasks, such as integration and differentiation. This narrow scope does not fully encompass the variety of mathematical problems encountered in educational settings, where issues of ambiguity and varying complexity levels are common.
3. Handling Ambiguity: Lample and Charton's work does not address the inherent ambiguity present in many mathematical problems, which can affect both the understanding and solving of such tasks. My project aims to incorporate mechanisms for classifying and managing ambiguous problem statements, enhancing the adaptability of AI-driven solutions in educational contexts.

2.2 Existing Math Problem Solvers

2.2.1 Wolfram Alpha and SymPy

Wolfram Alpha is recognized as an industry-standard computational engine that effectively performs symbolic math tasks, solves complex equations, and provides detailed step-by-step solutions. Its strengths lie in its ability to handle formal mathematical inputs with high precision, making it a powerful tool for users seeking exact computations in structured mathematical contexts. However, Wolfram Alpha faces challenges when dealing with natural language word problems. Its reliance on formal input formats limits its usability in educational settings, where problems are often presented in conversational language.

SymPy is a Python library designed for symbolic mathematics, enabling users to perform algebraic computations, calculus, and equation solving. Similar to Wolfram Alpha, SymPy provides exact answers and is particularly adept at managing structured mathematical problems. However, it requires problems to be formatted in symbolic notation, which poses a barrier for users who may not be familiar with the necessary syntax. This limitation highlights the need for a system that can bridge the gap between natural language inputs and the structured requirements of symbolic computation tools.

Both Wolfram Alpha and SymPy demonstrate exceptional accuracy in solving structured mathematical problems, but they share a significant weakness: their lack of natural language understanding restricts their ability to interpret and solve problems presented in everyday language. This limitation suggests a critical opportunity for integrating large language models (LLMs) with symbolic computation tools like SymPy. By leveraging LLMs, it becomes possible to develop systems capable of handling both structured and unstructured mathematical problems, enhancing their applicability in educational and practical settings.

2.2.2 MathQA: Towards Interpretable Math Word Problem Solving with Operation-Based Formalisms

MathQA introduces a novel approach to solving math word problems by employing category-based hierarchies for operation formalisms. This hierarchical structure categorizes math problems into broader groups, such as arithmetic, algebra, and geometry, with further subdivisions for specific operations like addition, subtraction, and exponentiation. This organization enables the model to recognize the overall type of problem as well as its subtasks, facilitating the application of the correct sequence of operations to derive a solution.

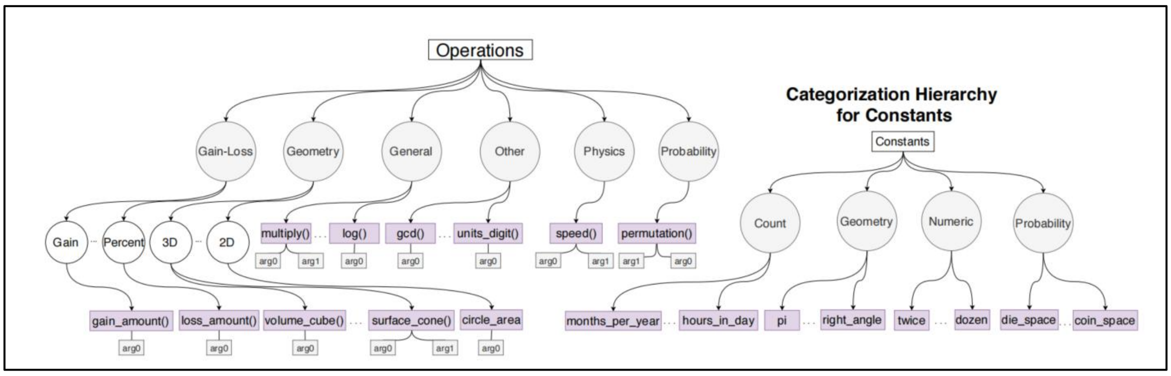


Figure 2.2. Category-based Hierarchies for Operation Formalism.

The architecture of MathQA follows a sequence-to-program model, translating math word problems into a structured series of operations. This transformation allows for the interpretability of math problems by decomposing each problem into manageable steps, using predefined templates to guide the solution process. For instance, a problem that requires multiplication followed by subtraction is represented as an ordered sequence of operations. The model leverages symbolic computation tools to execute these operations, effectively bridging the gap between natural language understanding and symbolic execution.

A key contribution of MathQA is its interpretable approach to problem-solving, which breaks down math word problems into operation-based formalisms. This structured methodology not only improves the model's ability to identify the specific operations required for a solution but also enhances the overall interpretability of the process. By utilizing hierarchical categorizations, MathQA presents an adaptable framework that can streamline problem classification and solution generation, making it particularly relevant in educational contexts.

Despite the strengths, there are notable gaps that can be addressed:

1. Limited Scope of Operations: MathQA's reliance on predefined templates may restrict its ability to handle unconventional or novel problem types that do not fit neatly into existing categories.
2. Natural Language Ambiguity: The approach may struggle with highly ambiguous natural language problems, where multiple interpretations could lead to different sequences of operations.
3. Adaptability to Diverse Domains: The current model primarily focuses on basic operations within specific categories, lacking the flexibility to incorporate multi-criteria classifications that consider problem complexity, ambiguity, and domain-specific requirements.

2.2.3 Solving Math Word Problem with Problem Type Classification

The Ensemble Math Word Problem Solver presents an innovative approach by implementing an ensemble-based strategy to tackle various types of math word problems. This model begins by categorizing problems into distinct types, such as arithmetic, algebra, geometry, and general word problems. Each category is assigned to a specialized solver tailored specifically to that problem type, while an overarching classifier directs each input to the most suitable model. This methodology enhances the accuracy and efficiency of the solving process, as it allows individual models to focus on their respective categories, thereby minimizing errors that might occur if a single model attempted to address all types of problems.

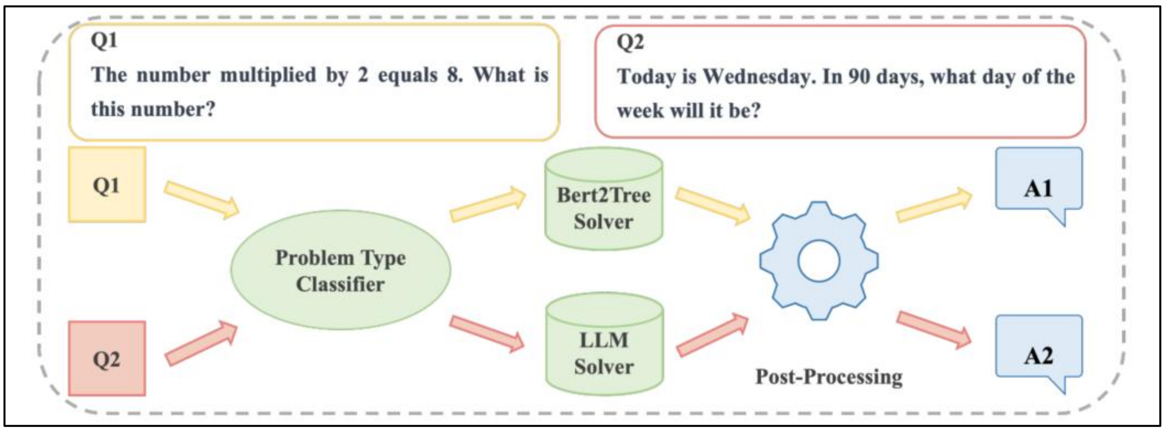

Q1
The number multiplied by 2 equals 8. What is this number?
Q2
Today is Wednesday. In 90 days, what day of the week will it be?
Q1
Q2
Problem Type Classifier
Bert2Tree Solver
LLM Solver
Post-Processing
A1
A2

Figure 2.3. Overview of Ensemble-MWP. The Problem Type Classifier assigns each MWP to either the Bert2Tree solver or the LLM solver based on a set of predefined rules. Once the classification is determined, the respective solver is employed to process the MWP and generate a preliminary result. The obtained result undergoes further Post-processing to derive the final answer.

The classification-based ensemble approach optimizes problem-solving by facilitating targeted strategies for specific mathematical tasks. For example, algebraic problems can utilize symbolic operations, whereas geometric problems can employ geometric-specific approaches. This division of labor not only reduces computational complexity but also improves performance across a wide range of mathematical topics. By enabling specialized models to handle nuanced aspects of different problem types, the ensemble framework aligns seamlessly with the goals of my project, which aims to implement a multi-criteria classification system for enhanced problem-solving.

A key contribution of this research lies in its demonstration of how effective classification can streamline the problem-solving process, allowing solvers to apply appropriate strategies based on the nature of the problem. This classification is essential for developing a robust expert system that can adapt to various domains, such as algebra and calculus, enhancing the overall performance and accuracy of mathematical problem-solving.

Despite the strengths, there are notable gaps that can be addressed:

1. Dependency on Individual Classifiers: The performance of the ensemble model heavily relies on the effectiveness of the individual classifiers. If any classifier underperforms, it can affect the overall efficiency of the ensemble.
2. Limited Integration of Contextual Information: The current ensemble framework may not adequately consider the contextual nuances of problems, which can lead to misclassifications or inappropriate solver assignments.
3. Scalability Challenges: As the number of categories increases, managing an ensemble of specialized models may become computationally intensive and complicate the training process.

2.2.4 SA-MEAS: SymPy-based Automated Mathematical Equations Analysis and Solver

The SymPy-Based System for the Visually Impaired (SA-MEAS) utilizes the powerful capabilities of SymPy to enhance accessibility in mathematical problem-solving for visually impaired individuals. This innovative system allows users to input mathematical equations via voice commands, which are then converted into symbolic representations by SymPy for processing. The ability to interact with the system without requiring visual engagement makes it particularly valuable for users with visual impairments. SA-MEAS outputs results in both audio and braille formats, ensuring that solutions are accessible and understandable.

One of the notable features of SA-MEAS is its capability to identify and analyze mathematical terms within algebraic expressions. By recognizing variables, constants, and coefficients, the system applies SymPy's symbolic operations to provide a detailed, step-by-step solution. This breakdown of complex equations not only aids in solving the problems but also enhances educational experiences by guiding users through each operation systematically. This feature is especially beneficial in learning environments, where understanding the progression of calculations is crucial for building mathematical comprehension.

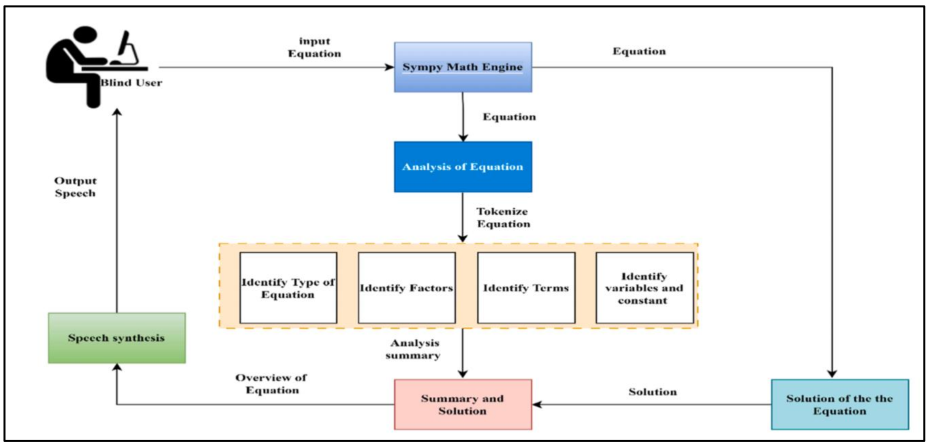


Fig. 2.4. Schematic architecture of the proposed solution.

The approach taken by SA-MEAS highlights the effectiveness of symbolic computation in generating accessible solutions. Integrating these capabilities could facilitate detailed explanations and improve user engagement with the symbolic computation aspects of an expert system.

Despite the strengths, there are notable gaps that can be addressed:

1. Limited Domain Scope: SA-MEAS primarily focuses on algebraic equations, which may limit its applicability in more advanced mathematical areas, such as calculus or differential equations.
2. Dependency on Voice Input: While voice commands enhance accessibility, reliance on this input method may pose challenges in noisy environments or for users who may have difficulty with speech recognition.
3. Step-by-Step Solutions: Although the system provides step-by-step solutions, it may not effectively convey the reasoning behind each step, potentially hindering deeper understanding for users.

2.3 Challenges in Word Problem Solving

2.3.1 A Comprehensive Review on Resolving Ambiguities in Natural Language Processing

This review provides a comprehensive examination of various types of ambiguities and their resolution techniques, which are essential for enhancing natural language processing (NLP) tasks. The discussion is centered around three primary categories of ambiguity: lexical ambiguity, where words possess multiple meanings; syntactic ambiguity, which arises from unclear sentence structures; and pragmatic ambiguity, which relies on context or external knowledge for correct interpretation.

Ambiguities pose significant challenges in the context of math word problems. For instance, terms such as "sum," "mean," and "difference" can be interpreted differently depending on their contextual use, leading to potential misunderstandings and errors in problem-solving. The paper explores various resolution techniques, including context-based disambiguation, rule-based parsing, and the application of pre-trained language models like BERT or RoBERTa. These models are adept at capturing contextual information, thereby improving the accuracy of complex sentence interpretation.

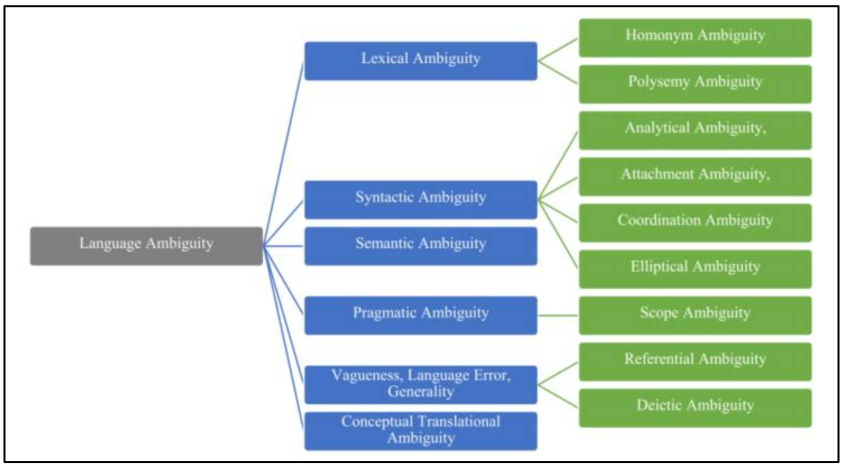


Figure 2.5. Types of ambiguity

Implementing similar ambiguity resolution methods prior to symbolic or numerical processing can greatly enhance the accuracy of any model in understanding user input, thereby mitigating the risks associated with misinterpretation. The insights provided in this paper about ambiguity types and resolution strategies are particularly valuable for the pre-processing of language within an expert system. They will aid in clarifying ambiguous terms and ensuring precise classifications before computation.

Despite the strengths, there are notable gaps that can be addressed:

1. Adaptation for Math-Specific Contexts: While the techniques mentioned are applicable to general NLP tasks, there may be a need for further adaptation specifically tailored to the nuances of math word problems, which often have unique linguistic structures and terminologies.
2. Limited Focus on Multimodal Inputs: The review primarily addresses text-based ambiguity resolution, neglecting the potential benefits of incorporating multimodal inputs (e.g., visual representations) that could enhance understanding of math problems.
3. Integration with Symbolic Processing: The paper does not explore how these ambiguity resolution techniques can be seamlessly integrated with symbolic computation methods, which is critical for creating a robust math problem-solving system.

2.3.2 Student Difficulties in Mathematizing Word Problems in Algebra

This study addresses the common challenges students encounter when translating algebraic word problems into symbolic form. It reveals that students often misunderstand relationships between variables or fail to grasp the significance of those variables. These misinterpretations are frequently attributed to ambiguous language in the problems or a lack of skills in structuring the problem effectively.

The findings underscore the importance of implementing guided solution pathways, which break problems down into individual steps. Such pathways encourage students to adopt a structured approach when forming equations, ultimately reinforcing their understanding. This insight is invaluable for designing educational systems aimed at scaffolding student learning, emphasizing transparency in the solution process.

The research emphasizes the significance of structured, step-by-step guidance in problem-solving. By incorporating similar guided pathways, it can support users in accurately translating word problems into symbolic equations, thereby reducing errors and improving overall accuracy.

The findings can inform the classification system by identifying common errors and obstacles students face when interpreting algebraic word problems.It highlights the necessity of a clear translation between natural language and symbolic expressions, which is essential for improving user experience in the expert system.

Despite the strengths, there are notable gaps that can be addressed:

1. Insufficient Focus on Diverse Learner Needs: While the study highlights common challenges faced by students, it does not account for the varying learning styles and needs of diverse learners, which could impact their ability to translate word problems into equations.
2. Limited Exploration of Technology Integration: The research does not investigate how technology or interactive tools can enhance guided solution pathways, potentially making the learning experience more engaging and effective.
3. Lack of Empirical Testing of Structured Approaches: The study advocates for structured problem-solving approaches but lacks empirical evidence demonstrating the effectiveness of these methods across different educational settings and demographics.

2.3.3 Automatic Classification of Question Difficulty Level: Teachers' Estimation vs. Students' Perception

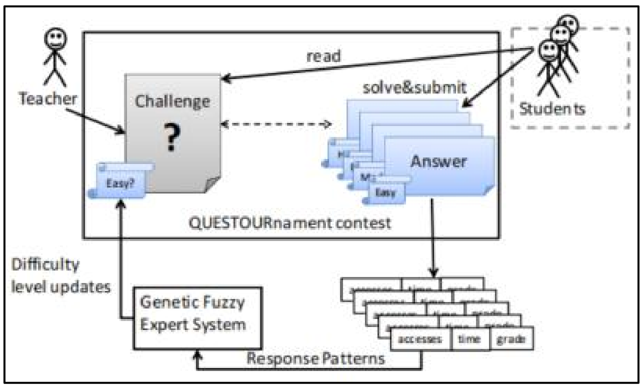


Figure 2.6. Diagram of the expert system for automatic classification of question difficulty

This paper presents an expert system designed for the automatic classification of question difficulty using machine learning techniques. By comparing these classifications to the perceptions of both teachers and students, it reveals a notable gap in how difficulty is perceived between these two groups. Specifically, teachers often rate the difficulty of math problems differently than students, highlighting the complexities involved in understanding question difficulty.

Understanding the discrepancy between teacher and student perceptions of difficulty could significantly inform the development of adaptive learning systems. By gaining insights into the student perspective, the project could adjust question difficulty in real time, tailoring it to better match user abilities and learning needs.

Integrating a difficulty classification system can enhance responsiveness to user skill levels. This capability would allow the system to offer tailored assessments and improve learning outcomes, ensuring that questions are aligned with the actual abilities of users.

The methods for difficulty classification discussed in the paper can be effectively integrated into a system to create adaptive test questions that cater to individual learner needs. Understanding the perception gap between teachers and students is crucial for improving overall learning outcomes, allowing the system to address the diverse challenges faced by learners in mathematical contexts.

Despite the strengths, there are notable gaps that can be addressed:

1. Limited Focus on Contextual Factors: The paper primarily addresses perceptions of difficulty without exploring how contextual factors—such as prior knowledge or the learning environment—might influence these perceptions.
2. Insufficient Integration of Real-Time Feedback: While the system aims to adapt to user abilities, it does not incorporate real-time feedback mechanisms to continuously refine difficulty classification based on ongoing performance.
3. Lack of Empirical Validation: The findings would benefit from empirical testing across diverse educational settings to establish the robustness of the automatic classification system.

2.4 Recent Advances in Mathematical Reasoning Using AI

2.4.1 MathCoder: Seamless Code Integration in LLMs for Enhanced Mathematical Reasoning

MathCoder presents a groundbreaking approach by integrating large language models (LLMs) with symbolic math solvers, specifically by generating SymPy-compatible code directly from math word problems expressed in natural language. The core of this system lies in its training on a robust dataset that pairs math problems with executable code, enabling it to effectively translate descriptive text into symbolic expressions and perform the necessary computations.

Dataset Creation and Fine-Tuning : MathCoder emphasizes the importance of generating training datasets by translating math problems into SymPy code. This process involves fine-tuning LLMs to convert text-based math problems into executable code, which significantly enhances mathematical reasoning and allows for efficient handling of symbolic operations.

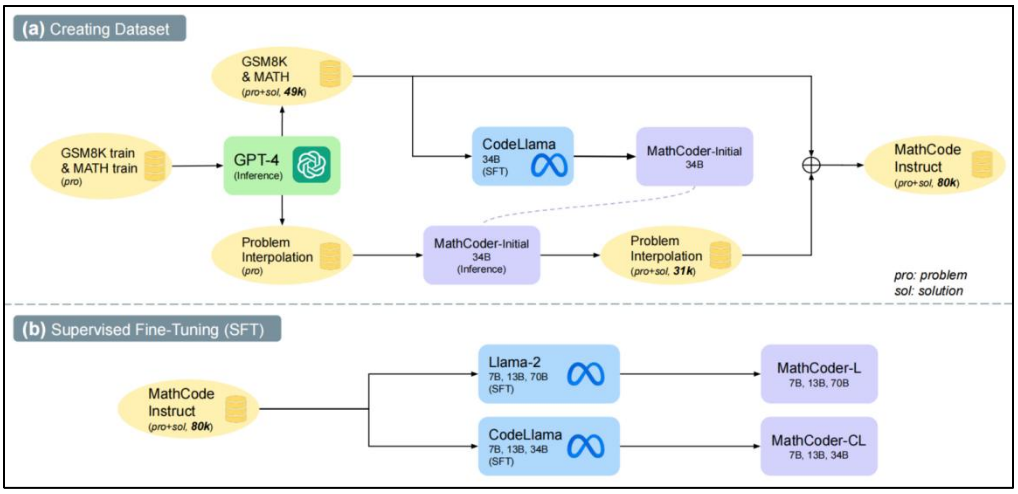

(a) Creating Dataset
GSM8K & MATH (pro+sol, 49k)
GSM8K train & MATH train (pro)
GPT-4 (Inference)
CodeLlama 34B (SFT)
MathCoder-Initial 34B
MathCode Instruct (pro+sol, 80k)
Problem Interpolation (pro)
MathCoder-Initial 34B (Inference)
Problem Interpolation (pro+sol, 31k)
pro: problem
sol: solution
(b) Supervised Fine-Tuning (SFT)
MathCode Instruct (pro+sol, 80k)
Llama-2 7B, 13B, 70B (SFT)
CodeLlama 7B, 13B, 34B (SFT)
MathCoder-L 7B, 13B, 70B
MathCoder-CL 7B, 13B, 34B

Figure 2.7. The process of dataset creation and model fine-tuning. (a) First, solutions for problems in the GSM8K and MATH datasets are collected from the GPT-4. Then they fine-tuned the CodeLlama-34B model on this data, producing the MathCoder-Initial. New problems are created using a novel prompt and their solutions are generated using MathCoder-Initial. (b) Finally, the new problems and solutions are combined with the existing training data to create the final dataset, which they used to fine-tune the base Llama-2 model, producing final MathCoder model.

The methodologies employed by MathCoder for creating symbolic datasets and training models align closely with the goals of my project. By effectively teaching LLMs to perform symbolic math operations, MathCoder exemplifies how code generation can serve as a bridge between natural language processing and symbolic math in educational contexts.

The dataset creation and fine-tuning techniques used in MathCoder could serve as a valuable source of inspiration for enhancing symbolic computation in my project. The ability to generate accurate SymPy code from natural language inputs will be crucial for developing an expert system capable of solving complex mathematical problems.

MathCoder effectively integrates LLMs with code generation techniques to address complex mathematical reasoning. By generating executable code from natural language inputs and utilizing symbolic computation tools, it presents a compelling model for transforming descriptive problems into solvable mathematical expressions.

This paper is highly pertinent to my thesis, as it illustrates the potential of LLMs in generating symbolic code. The insights gained from MathCoder's approach to code generation and correction can significantly enhance my expert system's capabilities in solving symbolic math problems.

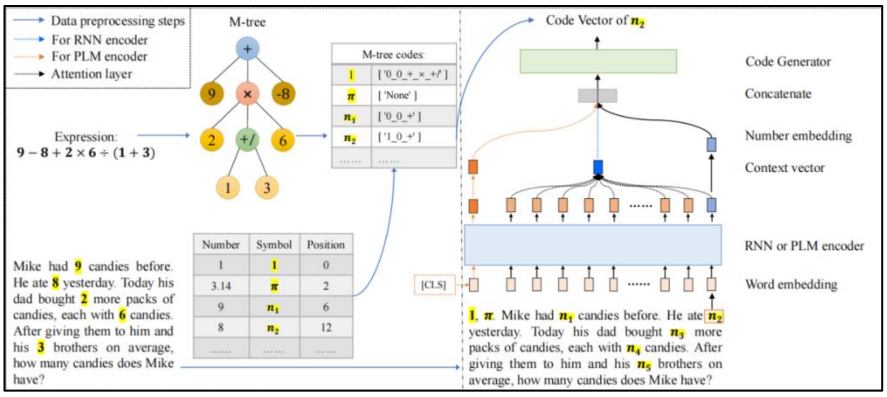
MathCoder provides an exemplary model for integrating code generation with symbolic computation, suggesting a feasible approach for implementation in my system. The ability of LLMs to enhance mathematical reasoning by converting text into runnable code underscores the potential for improving the accuracy and efficiency of problem-solving in educational settings.

Despite the strengths, there are notable gaps that can be addressed:

1. Complexity in Diverse Problem Types: While MathCoder is effective with a certain set of problems, further exploration is needed to assess its performance across a wider variety of mathematical disciplines and problem types.
2. Limited User Interaction Considerations: The current model lacks an emphasis on how user interactions (such as feedback loops) can enhance code generation accuracy and user experience.
3. Scalability Challenges: The system's approach to dataset creation and fine-tuning may present scalability issues when applied to larger and more diverse datasets.

2.4.2 Structure-Unified M-Tree Coding Solver for Math Word Problem

Structure-Unified M-Tree Coding Solver for Math Word Problems introduces the M-tree structure, which serves as a unified format for representing various math problems. Each node in the M-tree corresponds to a specific mathematical operation, allowing for the execution of multi-step operations in a coherent sequence. This approach utilizes a sequence-to-code model that translates these nodes into symbolic or executable code, enhancing the problem-solving capabilities of the system.



M-tree codes:	
1	['0_0_+_×_+/']
π	['None']
n_1	['0_0_+']
n_2	['1_0_+']
……	……

Number	Symbol	Position
1	1	0
3.14	π	2
9	n_1	6
8	n_2	12
……	……	……

Mike had 9 candies before. He ate 8 yesterday. Today his dad bought 2 more packs of candies, each with 6 candies. After giving them to him and his 3 brothers on average, how many candies does Mike have?

1, π. Mike had n_1 candies before. He ate n_2 yesterday. Today his dad bought n_3 more packs of candies, each with n_4 candies. After giving them to him and his n_5 brothers on average, how many candies does Mike have?

Figure 2.8. The left half is an example of MWP with the M-tree and M-tree codes, and the right half is the main architecture of our seq2code model

Tree Structure and Sequence-to-Code Model : The M-tree structure is pivotal in achieving high accuracy for complex math problems, particularly in scenarios requiring multi-step solutions. By representing diverse problem types in a sequence format, the M-tree enhances logical coherence in problem-solving. This structured representation allows for the clear translation of problem components into symbolic or executable code, making it a valuable asset for educational systems focused on teaching problem-solving strategies.

Despite its high accuracy in solving a range of complex math problems, the M-tree model faces challenges when addressing edge cases characterized by specific contextual requirements. This limitation underscores the importance of incorporating additional strategies to enhance context understanding, ensuring that the model can adapt to a wider array of problem scenarios.

The ability of the M-tree structure to represent various math problem types as sequences is a crucial insight for informing multi-step problem-solving strategies. By leveraging this unified representation, a system can enhance its problem classification capabilities and improve the accuracy of its solutions.The M-tree structure provides a cohesive framework for representing diverse mathematical expressions, facilitating the model's learning process and enabling it to solve different types of problems effectively. This contribution is significant for developing a comprehensive expert system that can manage a variety of mathematical formats and complexities.

The M-tree approach offers a flexible method for handling multiple problem formats. Its capability to unify diverse mathematical representations is particularly relevant for enhancing the system's performance in multi-step problem-solving contexts.

The M-tree structure provides a valuable solution for unifying diverse representations of math problems, which can significantly improve the classification and processing of mathematical problems. By enabling the execution of multi-step operations, the M-tree model enhances the overall efficacy of problem-solving strategies, aligning with the educational goals of fostering robust mathematical understanding.

Despite the strengths, there are notable gaps that can be addressed:

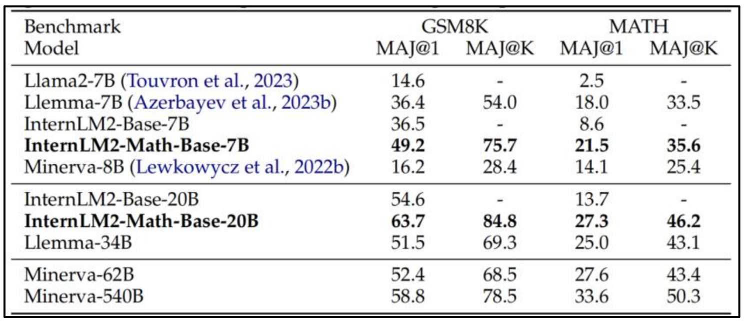
1. Contextual Limitations: While the M-tree model excels in handling standard multi-step problems, its performance may decline in edge cases where the context is highly specific or nuanced, indicating a need for further research on contextual adaptability.
2. Integration with Other Models: The literature lacks comprehensive studies on how the M-tree structure can be integrated with existing LLMs or symbolic computation frameworks for enhanced problem-solving efficiency.
3. Scalability and Complexity: Future studies should address the scalability of the M-tree model when applied to an extensive range of math problems, particularly in educational contexts.

2.4.3 InternLM-Math

InternLM-Math introduces innovative techniques such as reward modeling, data augmentation, and Chain-of-Thought (CoT) reasoning to enhance the processing and solving of complex math word problems. Each of these components contributes to the model's overall effectiveness and aligns closely with the goals of developing a comprehensive math-solving system.

InternLM-Math employs reward modeling to improve model accuracy by refining response quality through feedback loops. Data augmentation broadens the model's generalizability, allowing it to perform well across a wider range of problems. CoT reasoning enhances the model's ability to solve multi-step problems by decomposing them into manageable parts, thereby fostering a clearer understanding of each solution step. This systematic approach aligns with my project's objective of creating an effective math-solving system capable of tackling various problem types.

The combination of these techniques makes InternLM-Math highly effective for processing complex word problems. The integration of CoT reasoning helps structure solutions clearly, allowing users to follow along and understand the problem-solving process. This not only improves the robustness of the solutions but also enhances generalizability across different types of mathematical problems.



Benchmark	GSM8K		MATH	
Model	MAJ@1	MAJ@K	MAJ@1	MAJ@K
Llama2-7B (Touvron et al., 2023)	14.6	-	2.5	-
Llemma-7B (Azerbayev et al., 2023b)	36.4	54.0	18.0	33.5
InternLM2-Base-7B	36.5	-	8.6	-
InternLM2-Math-Base-7B	**49.2**	**75.7**	**21.5**	**35.6**
Minerva-8B (Lewkowycz et al., 2022b)	16.2	28.4	14.1	25.4
InternLM2-Base-20B	54.6	-	13.7	-
InternLM2-Math-Base-20B	**63.7**	**84.8**	**27.3**	**46.2**
Llemma-34B	51.5	69.3	25.0	43.1
Minerva-62B	52.4	68.5	27.6	43.4
Minerva-540B	58.8	78.5	33.6	50.3

Table 2.3. Compare pre-trained models using ICL. The metric is majority voting accuracy. K = 100 for the GSM8K benchmark and K = 256 for the MATH benchmark. They used greedy decoding when K = 1. They sampled their models using a temperature of 0.7 when K > 1.

The use of CoT reasoning and data augmentation techniques in InternLM-Math provides valuable insights for handling complex, multi-step problems in my project. These methodologies can significantly enhance the user experience by fostering a deeper understanding of problem-solving processes. InternLM-Math represents a significant advancement in the realm of math problem-solving by combining language models with symbolic reasoning tools such as SymPy and formal proof systems. This integration allows for efficient code generation, symbolic execution, and proof verification, marking a notable evolution in educational tools for math instruction.

The methodologies employed in InternLM-Math align closely with my system's goals, particularly regarding the ability to handle a wide array of math problems, from basic word problems to intricate calculus equations. This relevance is crucial as I aim to develop an adaptable and comprehensive solution for math problem-solving.

InternLM-Math provides an advanced approach by effectively combining LLMs with symbolic computation, demonstrating a pathway for enhancing math problem-solving capabilities. The introduction of Python code execution for verifying mathematical solutions could significantly improve the robustness and reliability of my system, ensuring that solutions are not only generated but also validated for accuracy.

Despite the strengths, there are notable gaps that can be addressed:

1. Integration of Techniques: There is limited exploration on how effectively these techniques can be integrated with existing educational tools and platforms for optimal impact on student learning outcomes.
2. Scalability of Techniques: The scalability of reward modeling and CoT reasoning in varied educational settings remains under-researched, especially concerning their effectiveness with different age groups and skill levels.
3. Comparative Analysis: More studies are needed to compare the effectiveness of InternLM-Math with other LLMs in handling diverse problem types and complexities.

2.5 Student Learning and Problem Complexity

2.5.1 Subject Classification and Difficulty Ranking of Math Problems

The study shows the comparison of LSTM and BERT for Classification focuses on the effectiveness of LSTM (Long Short-Term Memory) networks versus BERT (Bidirectional Encoder Representations from Transformers) in classifying math problems by subject type and difficulty and reveals that BERT, with its advanced text-processing capabilities, surpasses LSTM in managing complex and ambiguous language, which is critical for accurately classifying math problems based on language cues and difficulty levels.

Utilizing BERT for the classification of math problems enhances the project's capacity to assess complexity and ambiguity effectively. BERT's superior handling of nuanced language allows for a more precise categorization of math problems, enabling the model to better identify challenging word problems. This capability is essential for adapting the educational experience to meet the needs of students.

The key takeaway is that adopting a BERT-based approach for classification can significantly improve the system's ability to process language-intensive math problems with greater accuracy. This improvement is crucial for creating an educational tool that effectively supports students in tackling complex mathematical concepts.
Subject Classification and Difficulty Ranking of Math Problems. The Key Contribution of this research is its proposal for classifying math problems by difficulty and ranking them based on complexity metrics. This methodology is essential for adapting tests and learning environments to align with students' skill levels, ensuring that the educational materials provided are appropriate for their abilities.

The findings from this research will aid in creating effective difficulty metrics that guide the problem classification process in my system. By leveraging these metrics, my project can ensure that the mathematical challenges presented to students align with their learning needs and capabilities, ultimately enhancing their understanding and problem-solving skills.

Despite the strengths, there are notable gaps that can be addressed:

1. Integration of Different Models: While BERT outperforms LSTM, there is limited research on the potential benefits of integrating both models for improved classification outcomes, especially in educational contexts.
2. Generalizability Across Domains: The effectiveness of BERT for classifying math problems across various educational settings and problem types has not been extensively studied.
3. Evaluation Metrics: More emphasis is needed on developing robust evaluation metrics for assessing the performance of these models in educational applications.

2.5.2 Solving Quantitative Reasoning Problems with Language Models

Minerva Model with Code Generation focuses on the model's ability to translate quantitative reasoning problems into executable code, which enhances its capacity to tackle advanced reasoning tasks. By converting natural language input into precise calculations, Minerva effectively addresses complex quantitative reasoning problems, showcasing the power of integrating code generation with natural language processing (NLP).

Minerva's strong performance on challenging benchmarks like the MATH and MMLU-STEM datasets highlights its adaptability to reasoning tasks. This success underscores the significance of code generation in enhancing the problem-solving abilities of language models.

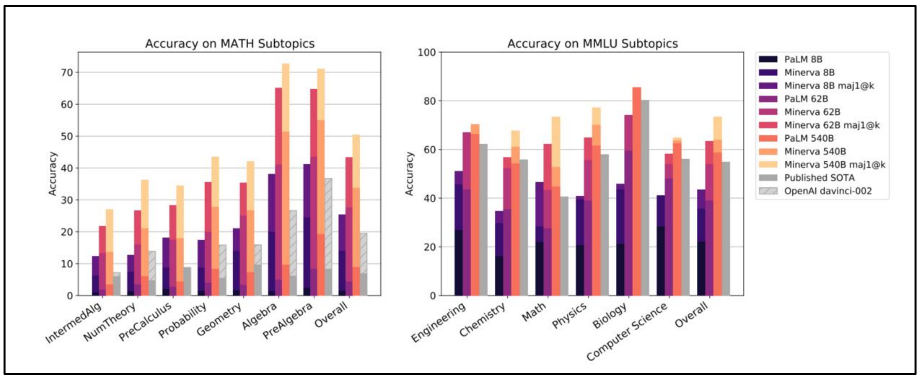


Figure 2.9. Performance on MATH and MMLU-STEM by subtopic in comparison with other models.

Minerva’s ability to generate code from natural language input provides a valuable model for integrating NLP with symbolic computation. This integration can enhance quantitative reasoning and improve the system’s capability to solve multi-step problems effectively.

This paper presents methods for leveraging LLMs to solve quantitative reasoning problems, demonstrating how these models can effectively handle reasoning-intensive tasks common in math word problems. The key contribution of this research is its demonstration of the effectiveness of LLMs in solving multi-step reasoning problems. The insights gained can inform the design and functionality of an expert system, particularly in its ability to tackle complex mathematical reasoning tasks.

This study provides valuable techniques and methodologies that could significantly improve the reasoning capabilities, particularly in addressing multi-step problems.

Fine-tuning LLMs is essential for effectively handling multi-step reasoning in math word problems. This requirement aligns with my system's goals of addressing complex mathematical problems, ensuring that the solutions generated are accurate and contextually relevant.

Despite the strengths, there are notable gaps that can be addressed:

1. Integration of Different Techniques: There is a need for more research on how combining different models and techniques can further enhance the performance of systems like Minerva in various mathematical contexts.
2. Generalization of Capabilities: While Minerva demonstrates proficiency on specific datasets, its performance across a broader range of problem types and domains requires further exploration.
3. User-Centric Design: The current literature does not sufficiently address how to design systems that effectively guide users through complex reasoning tasks while leveraging LLMs.

2.6 LILA dataset (A Unified Benchmark for Mathematical Reasoning)

This dataset offers an extensive, structured benchmark specifically aimed at evaluating and enhancing AI models' mathematical reasoning capabilities. This dataset was meticulously designed to cover a broad spectrum of mathematical tasks, ranging from basic arithmetic to advanced calculus, and includes over 133,000 problems in multiple formats, like question-answering, fill-in-the-blank, and reading comprehension.

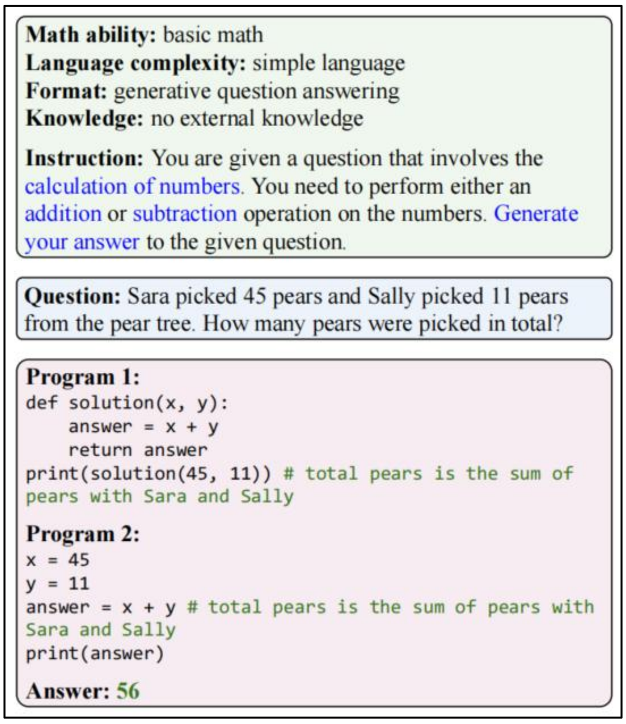
It is built on the integration of 20 existing mathematical reasoning datasets, enriched with four unique annotations to better address problem complexity and improve model interpretation:

1. Mathematical Ability: Identifies areas such as algebra, geometry, and calculus, providing an easy way to assess a model’s strengths or weaknesses across disciplines.
2. Language Complexity: Classifies problems as simple, complex, or language-independent, which can help in gauging models’ ability to understand and process diverse language structures.
3. Question Format: Differentiates question types, including multiple-choice and generative Q&A, to evaluate adaptability across problem formats.
4. External Knowledge: Annotates questions requiring common sense or additional scientific knowledge, helping assess how well a model leverages contextual knowledge in mathematical problem-solving.

One of LILA's standout features is the inclusion of **annotated Python programs for each question**. These programs not only produce the correct answers but also provide a structured reasoning process, supporting models to generate solutions with traceable logic. Additionally, LILA is equipped with evaluation subsets like **LILA-OOD** (out-of-distribution) and **LILA-ROBUST** (linguistic perturbations) to test generalization and resilience against changes in language and format, both of which are key factors in enhancing model robustness.

LILA's creators also introduced **BHĀSKARA**, a model fine-tuned specifically on this dataset, achieving a substantial improvement in handling varied tasks. The multi-task approach of BHĀSKARA shows a 21.83% improvement in F1 scores over single-task models, emphasizing the importance of diverse data for training adaptable AI models.

The LILA dataset is ideal for projects focused on comprehensive mathematical reasoning, given its multi-dimensional structure and wide range of problem types. For my own project, LILA's Python-annotated reasoning processes and varied question formats will be essential in training models to not only solve math problems accurately but also explain the solution path. This dataset provides an opportunity to create models that can handle real-world variations in problem structure, potentially leading to more reliable and interpretable AI-based problem solvers.


Math ability: basic math
Language complexity: simple language
Format: generative question answering
Knowledge: no external knowledge

Instruction: You are given a question that involves the calculation of numbers. You need to perform either an addition or subtraction operation on the numbers. Generate your answer to the given question.

Question: Sara picked 45 pears and Sally picked 11 pears from the pear tree. How many pears were picked in total?

Program 1:
```
def solution(x, y):
    answer = x + y
    return answer
print(solution(45, 11)) # total pears is the sum of
pears with Sara and Sally
```

Program 2:
```
x = 45
y = 11
answer = x + y # total pears is the sum of pears with
Sara and Sally
print(answer)
```

Answer: 56

Figure 2.10. A data example with two Python programs in LILA. One program annotation uses function construct whereas the other one is a plain script without function. The instruction for each task and categories across four dimensions are annotated for developing LILA.

2.7 Conclusion

This literature review underscores the significant advancements in the integration of large language models (LLMs) and symbolic computation tools for solving mathematical problems, particularly within educational contexts. The findings across various studies reveal the critical challenges that students face when translating natural language into symbolic expressions, including issues of misinterpretation and ambiguity. Several models, such as MathCoder, InternLM-Math, and Minerva, illustrate the effectiveness of combining code generation with structured problem-solving approaches to enhance mathematical reasoning capabilities.

The exploration of techniques like the M-tree structure and Chain-of-Thought reasoning highlights the necessity of providing structured pathways that guide users through complex problem-solving processes. Moreover, the comparative analysis of LSTM and BERT illustrates the advantages of utilizing sophisticated models that are adept at processing the intricacies of mathematical language.

Additionally, understanding the discrepancies in perceived difficulty between teachers and students is pivotal for developing adaptive learning systems. By incorporating user feedback and adaptive assessment methodologies, the expert system being developed can better align with individual learning needs, thereby enhancing user engagement and learning outcomes.

In conclusion, the insights drawn from this review significantly inform the design and development of my expert system aimed at addressing mathematical problems across various domains. By leveraging advanced LLMs, integrating structured solution pathways, and tackling user-specific challenges, this project seeks to improve both the accuracy and efficacy of mathematical problem-solving while fostering a supportive educational environment. The combination of these methodologies positions the system to make a meaningful contribution to both educational and practical applications of mathematics.

3. Problem Definition

3.1 Problem Formulation

One of the main goals in math education and problem-solving systems is to build a tool that can understand and solve a variety of math problems, especially word problems. While AI has made great progress with understanding language and doing symbolic math separately, there's still a big challenge in combining these two abilities to solve the mathematical problems effectively. Most AI systems struggle when math word problems are given in natural language because they need to figure out the right math operations from the text, and then solve the problem accurately.

The objective of this thesis is to create an expert system that uses large language models (Llama, Mistral etc.) and symbolic computation tools (like SymPy) to solve math problems. The system will focus on classifying math problems based on their complexity, ambiguity, type, and number of operations or functions etc. and also the steps to solve the problem. Once classified, the system will determine how to solve the problem—whether it can be handled directly by the language model or if more detailed symbolic math tools are needed.

3.2 Key Challenges

3.2.1 Ambiguity in Natural Language

A big challenge when solving math word problems is the ambiguity in the way questions are phrased. Sometimes, the same words can mean different things depending on the context, which makes it difficult for AI systems to figure out what exactly they need to calculate.

For example, if a problem says, "John has twice as many apples as Sarah, what is the total number of apples." Here the problem didn't mention the number of apples Sarah has. If the wording is unclear, the system might not be able to apply the any proper operation and it may hallucinate. So, one of the main goals is to handle and resolve these ambiguities before starting the math operations and if there is any ambiguity detected then the model will check with the user again before proceeding with the operation.

3.2.2 Multi-Step Reasoning and Symbolic Computation

Many math problems are not solved in just one step. Instead, they require multi-step reasoning, where several operations need to be performed in sequence.

For example, a problem might ask to first calculate the area of a shape, then add or subtract values, and finally give a final result or for some other problem, like finding the curl, it may ask us to identify the components of vector field, then finding partial derivative differences and output the answer.

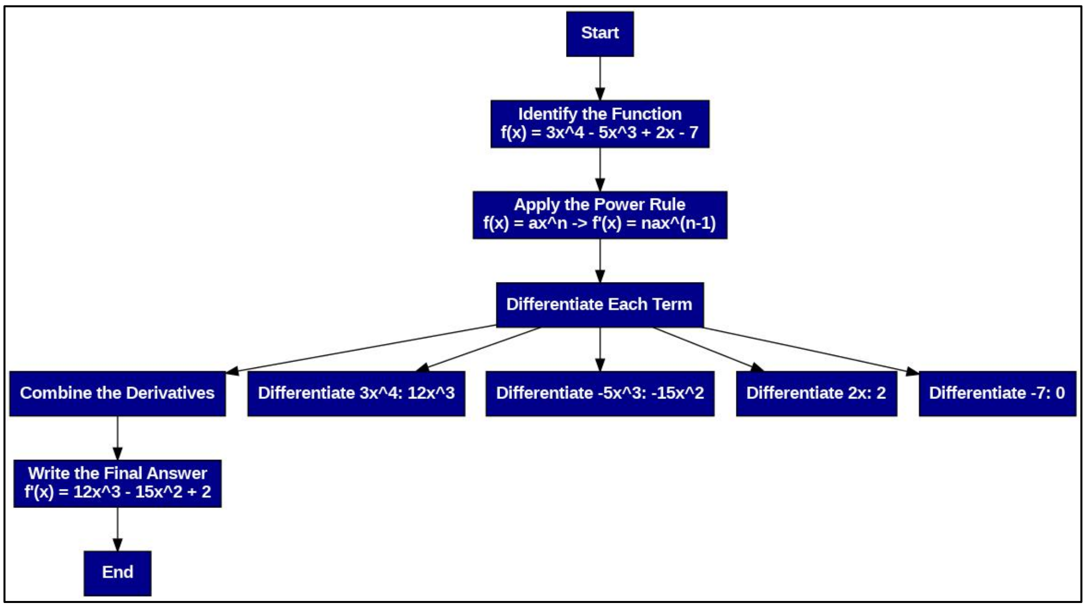


Figure 2.11. Example of steps involved in solving a calculus problem.

The challenge here is that the system must break down the problem into smaller steps, perform the math for each step, and keep track of everything correctly. For this, the system will combine classification part which will also provide the relevant steps rather than the LLM doing everything to understand the problem, which can later be passed to either LLM to solve the problem or SymPy to solve the math.

3.2.3 Complexity of Mathematical Problems

Math problems vary in complexity. Some problems are simple, like solving a basic equation, while others are more complex, involving calculus or multiple variables. The system needs to evaluate how complex a problem is, which will help it choose the right method to solve it.

For example, solving a basic algebraic equation is much simpler than calculating a multivariable integral. The system should be able to tell the difference between these types of problems and adjust its approach accordingly. This parameter will also become the decision point to pass to euther LLM or SymPy.

3.2.4 Exact vs. Approximate Solutions

Some math problems need an exact solution (such as when solving for an exact number), while others may allow for an approximate solution (such as estimating a value). The system must know when it’s appropriate to give an exact answer versus an estimate. This is especially important for complex functions where exact solutions might be too hard to compute, and an approximation is good enough.

For example, calculating the integral of a complex function might require using numerical methods if an exact solution isn’t possible.

3.2.5 Domain-Specific Challenges

Math problems come from different domains, such as algebra, geometry, calculus etc. Each domain has its own set of rules and operations, which makes it important for the system to recognize what type of problem it’s dealing with and apply the right methods.

For example, geometry problems often require calculations involving shapes and areas, while algebra problems are more about solving for variables. The system must first classify the problem by domain and then apply the correct solution strategy.

3.3 Multi-Criteria Classification System

The goal of this thesis is to develop a system that can classify math problems based on several important factors:

1. Ambiguity: Detecting and clarifying ambiguous terms or phrases in word problems
2. Domain: Classifying the problem based on the domain (e.g., algebra, geometry, calculus).

3. Complexity: Evaluating how complex the problem is, based on the number of steps, operations, or functions involved.
4. Solution Type: Deciding whether the problem requires an exact solution or an approximate solution.
5. Number of Operations and Functions: Counting and identifying the math operations (addition, multiplication) and functions (logarithms, trigonometry) used in the problem.
6. Steps to solve the problem: Manual plain word statements, without any calculation to guide the LLM to solve the problem.

By classifying the problems in this way, the system can determine how to approach solving each problem efficiently. It will also ensure that multi-step problems, ambiguous problems, and complex problems are handled correctly, improving both accuracy and speed by passing it to relevant solvers. For example, if a problem is not that complex then Llama itself can solve it and that will save the processing time for simpler problems.

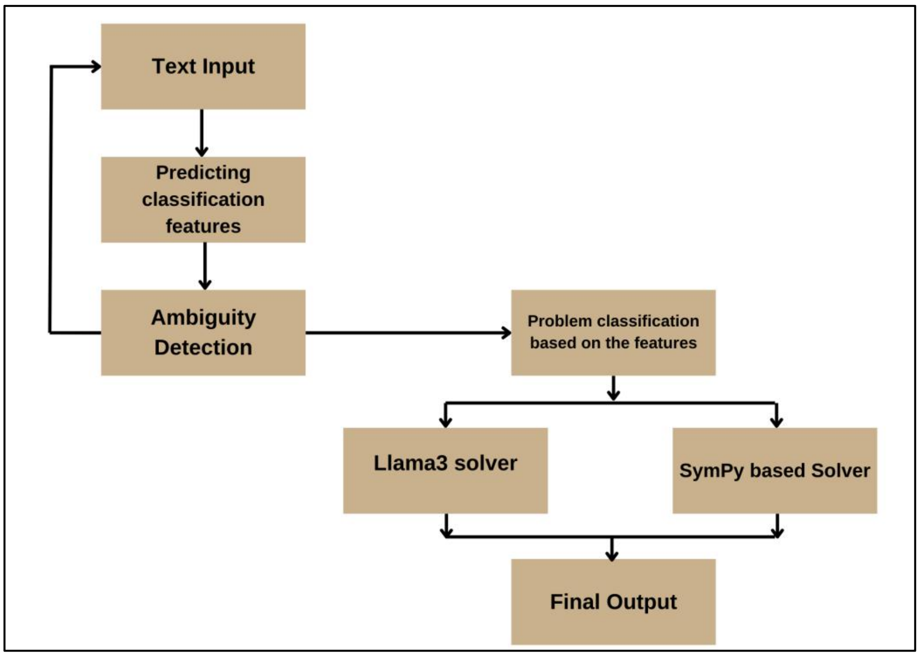

Text Input
Predicting classification features
Ambiguity Detection
Problem classification based on the features
Llama3 solver
SymPy based Solver
Final Output

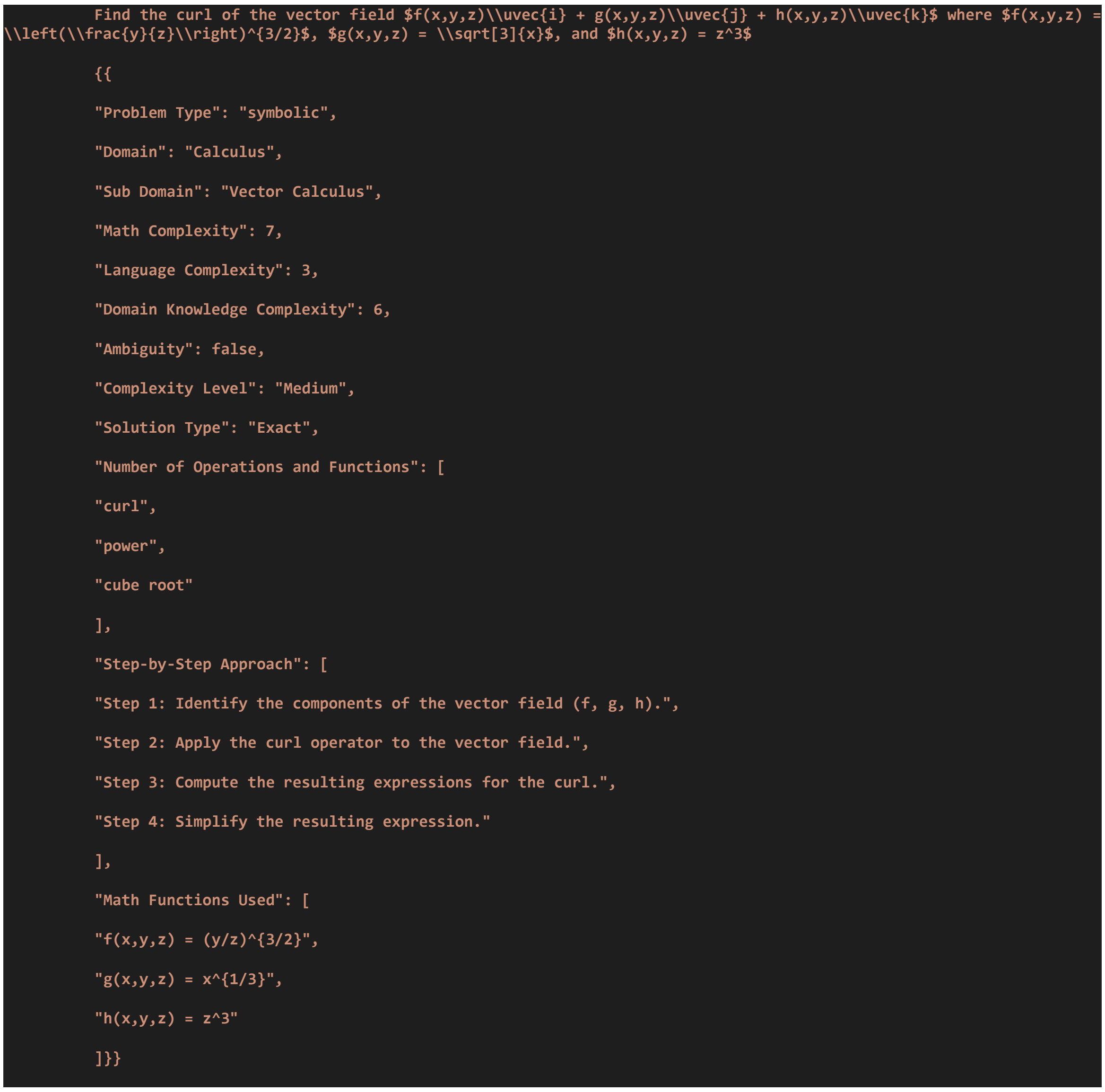
Figure 2.12. Flowchart of the whole approach.

4. Initial and Preliminary Approach

In the initial phase of developing my math problem-solving expert system, my goal was to create a workflow that could handle math word problems in natural language, generate corresponding symbolic code, correct errors iteratively, and provide accurate results. This first approach helped me explore the system's basic capabilities and understand the potential areas for improvement.

My early system followed a straightforward workflow to tackle each problem:

1. Problem Input: The process began with a math word problem presented in natural language. This allowed me to test the system's ability to interpret everyday language descriptions of math tasks.
2. Code Generation by LLM: The problem was passed to a language model, which generated symbolic code based on the input. For this, I used SymPy to handle symbolic operations, aiming for exact calculations in outputs.
3. Error Correction Loop: Since initial code outputs were not always correct, I set up an error-checking loop. If the generated code had syntax or logic errors, the system would capture these, provide feedback to the model, and prompt it to correct the code until a working solution was reached.
4. Execution and Output: Once the code was error-free, it was executed to yield a final answer. Where possible, I compared this output against expected results to gauge accuracy.
5. Augmented LILA Dataset: I started augmenting one of the json files of calculus where I used Llama3 to generate the different classification features for some problems.



```
        Find the curl of the vector field $f(x,y,z)\\uvec{i} + g(x,y,z)\\uvec{j} + h(x,y,z)\\uvec{k}$ where $f(x,y,z) =
\\left(\\frac{y}{z}\\right)^{3/2}$, $g(x,y,z) = \\sqrt[3]{x}$, and $h(x,y,z) = z^3$

        {{

        "Problem Type": "symbolic",

        "Domain": "Calculus",

        "Sub Domain": "Vector Calculus",

        "Math Complexity": 7,

        "Language Complexity": 3,

        "Domain Knowledge Complexity": 6,

        "Ambiguity": false,

        "Complexity Level": "Medium",

        "Solution Type": "Exact",

        "Number of Operations and Functions": [

        "curl",

        "power",

        "cube root"

        ],

        "Step-by-Step Approach": [

        "Step 1: Identify the components of the vector field (f, g, h).",

        "Step 2: Apply the curl operator to the vector field.",

        "Step 3: Compute the resulting expressions for the curl.",

        "Step 4: Simplify the resulting expression."

        ],

        "Math Functions Used": [

        "f(x,y,z) = (y/z)^{3/2}",

        "g(x,y,z) = x^{1/3}",

        "h(x,y,z) = z^3"

        ]}}
```

Figure 2.13. One example classification of a problem in JSON format.

This workflow provided a baseline to measure the performance of the language model in both understanding and executing math problems from natural language.

5. Preliminary Results

My initial experiments revealed both promising outcomes and areas where improvements are needed. Below are the key findings:

1. Code Generation Accuracy: The model generated correct code on the first attempt for simpler problems, such as basic arithmetic operations (e.g., addition, subtraction) or straightforward algebra. For multi-step or complex operations (like calculus problems involving differentiation or integration), the model often failed to produce error-free code in a single try, highlighting a need for enhanced understanding and code accuracy.
2. Effectiveness of the Error Correction Loop: The error correction loop was helpful in catching and addressing simple syntax errors. In several cases, the model could correct these errors and refine the code successfully after one or two iterations. However, this process added significant processing time for more complex problems, as multiple rounds of correction were often required. For complex problems, each iteration increased overall execution time, affecting the system's efficiency.
3. Processing Speed and Bottlenecks: The iterative nature of the error correction process contributed to longer processing times, with execution times for simpler problems averaging around 10 seconds but sometimes extending to 20 seconds or more for complex issues. Long processing times emerged as a bottleneck in cases requiring multiple iterations to resolve errors, indicating a need for optimizing the correction loop to handle complex problems faster.
4. Accuracy Across Different Problem Types: The system was effective for direct, straightforward problems (e.g., solving single-variable algebra equations), demonstrating high accuracy in these cases. However, ambiguity in word problems—especially those requiring contextual understanding (like "twice as many")—often led to logical errors in code generation. This points to the challenge of interpreting natural language with complex dependencies accurately.
5. Observations on Problem Types and Complexity: For problems that required basic operations, the model was able to classify and solve them with ease. More complex, layered problems involving multiple steps or domain knowledge (e.g., real-world scenarios) presented challenges. These insights suggested that categorizing problems by type and complexity could enhance the accuracy and efficiency of the solution process.
6. The Llama model was generating the output in improper order and that could be due to many factors, like prompt structure or model's internal configuration etc. So will alkso work on enhancing that prompt structure for creating the dataset or maybe will consider using chatGPT for dataset generation.

The insights gained from these early experiments will help me in the future steps:

1. Refinement of Code Generation: Improving the initial prompts or incorporating additional training data focused on math-related tasks could help the model generate more accurate code on the first attempt, particularly for complex mathematical functions.

2. Optimization of Error Correction: Streamlining the error-correction process to reduce the number of iterations needed could greatly improve processing time and efficiency. I aim to introduce more targeted feedback mechanisms for common syntax or logic errors.
3. Multi-Criteria Problem Classification: Implementing a classification system on the augmented dataset using Llama3 on LILA that categorizes each problem based on type, complexity, required operations etc. will allow the model to adjust its approach to match the problem's needs. This should also help prioritize simpler problems, saving time and resources for more complex issues.

These preliminary results confirm the value of combining symbolic computation tools with language models and highlight the need for refinement in both code generation and problem classification.

6. Plan for Stage 2

For **Phase 2**, the focus will be on building a robust classification and solving system for math problems by refining the approach from Phase 1, expanding the dataset, and implementing additional metrics.

Following are the targets for Phase 2:

1. I will **augment the entire dataset** using the Llama3 model to prepare for training the classification model.
2. For **classification model development**, I am developing a multi-criteria classification model that will categorize math problems based on complexity, type, domain, ambiguity, and solution type. This model will guide the solving approach, enhancing the system's efficiency and accuracy. I plan to train the model on the augmented dataset created on LILA and consider using a recommendation engine for this purpose.
3. I aim to implement **algorithmic enhancements** for error correction and code generation. This includes:
    a) Developing an error prediction model that identifies common errors in code generation, which will reduce the need for iterative corrections.
    b) Creating and testing more focused prompts that encourage LLMs to produce cleaner, more accurate code on the first attempt, especially for multi-step problems.
    c) Integrating refined checks with SymPy to verify that the generated code adheres to the expected mathematical logic for each problem type.
4. I will also focus on the **development of complexity and difficulty metrics**. My objectives include:
    a) Creating a system for rating each problem's complexity based on factors such as the number of operations involved (e.g., simple addition vs. multi-step calculus) and types of functions (e.g., basic arithmetic vs. trigonometric or logarithmic).
    b) Setting up a model to predict how difficult each problem might be for different student levels, based on parameters like language complexity, number of steps, and ambiguity.
    c) Conducting validation testing to cross-check difficulty ratings against common benchmarks in educational testing.

5. For **evaluation metrics and testing**, I will define robust evaluation metrics to measure the accuracy and efficiency of the model's classification and problem-solving capabilities. This will involve:

a) Calculating accuracy based on the correct classification of problem type, complexity, and domain.
b) Tracking error rates in initial code generation and measuring the effectiveness of the error correction loop in reducing these rates.
c) Measuring processing time from input to final solution, focusing on minimizing delays in the correction loop and maximizing efficiency.
d) Implementing a framework for testing the model on a validation dataset and recording performance across all metrics.

6. I will develop an **enhanced dataset** for classification and solution testing that includes additional classification labels. This will involve:

a) Creating a dataset where each problem includes comprehensive labels for classification, such as type, complexity, solution type, and ambiguity.
b) Designing specific test cases to evaluate the model's handling of different problem types, particularly multi-step and ambiguous problems.
c) Curating a subset of edge cases known to challenge traditional solvers to see how the system manages these cases.

7. In my **final report documentation**, I will document the findings and progress in a detailed report, using visuals to enhance understanding and clarity. I plan to:

a) Clearly outline the improvements made in Phase 2, including dataset expansion, classification criteria, error correction refinements, and evaluation metrics.
b) Include diagrams and tables that summarize key data points, metrics, and improvements from Phase 1 to Phase 2.

7. Analysis of work in this field (Scopus)

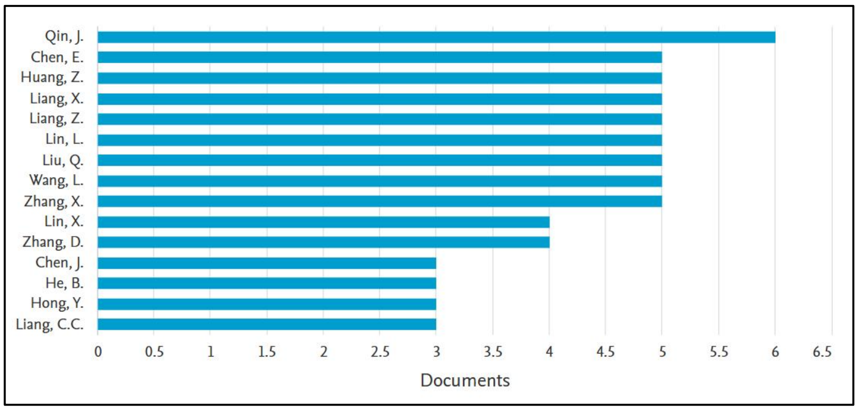


Figure 7.1. Top 15 authors who have worked in this field.

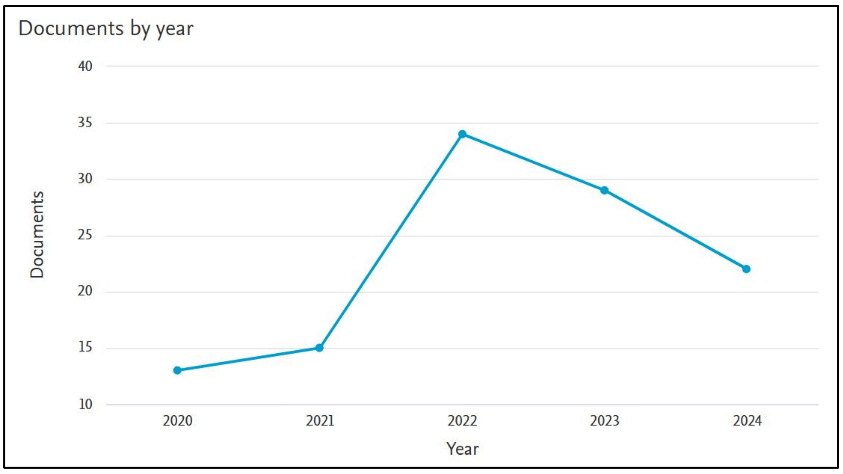


Figure 7.2. Number of papers in a year in this field.

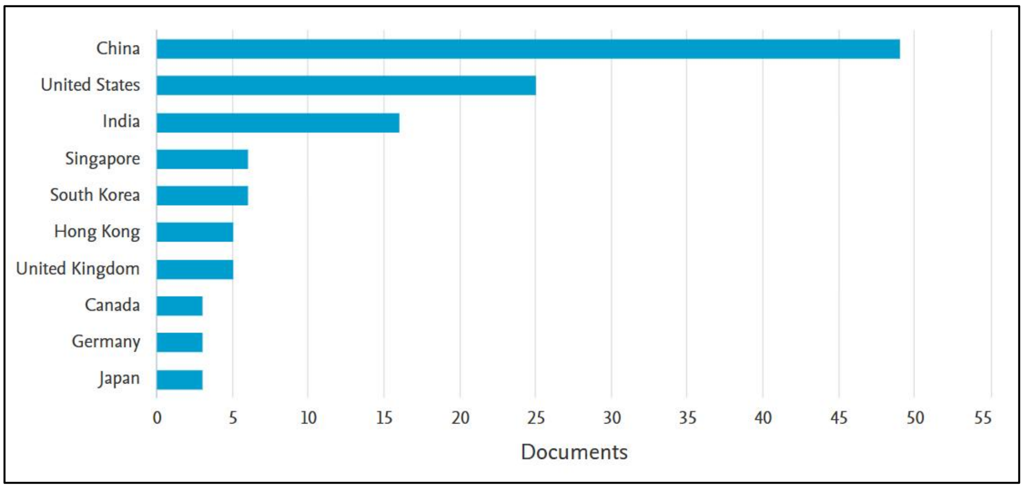


Figure 7.3. World Graph in this field.

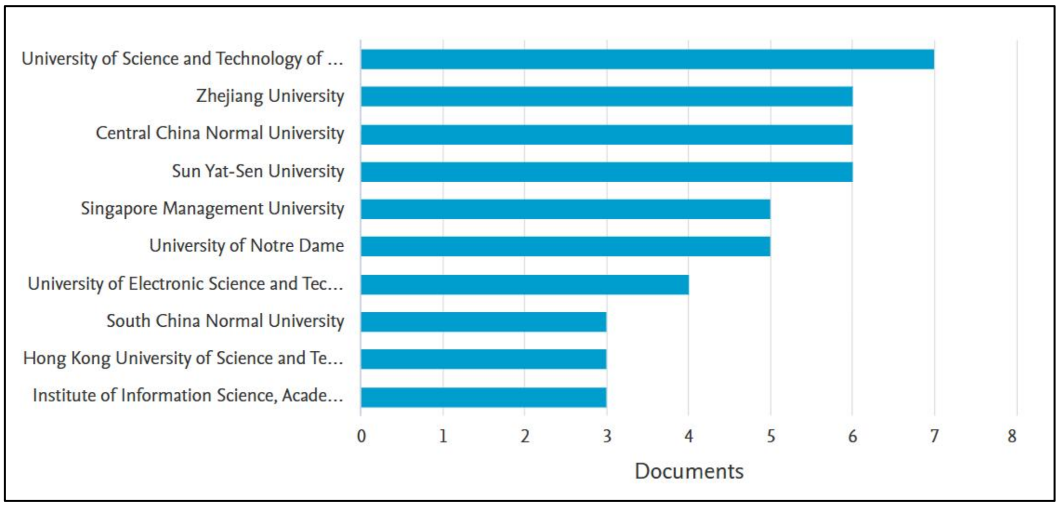


Figure 7.4. University counts in this field.

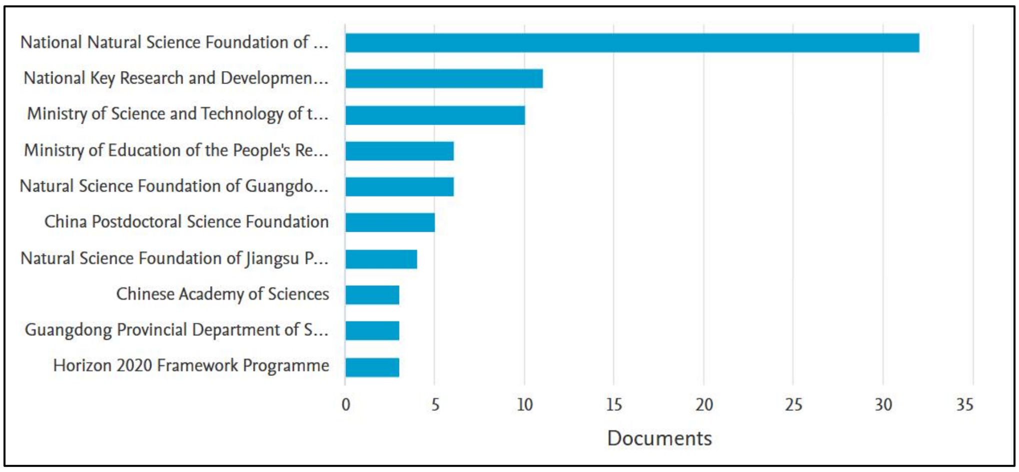


Figure 7.5. Funding Agencies in this field.

8. References

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv preprint arXiv:1907.11692*. https://arxiv.org/abs/1907.11692

Lample, G., & Charton, F. (2019). Deep Learning for Symbolic Mathematics. *arXiv preprint arXiv:1912.01412*. https://arxiv.org/abs/1912.01412

Zhao, W., Li, L., Bansal, M., & Li, W. (2019). MathQA: Towards Interpretable Math Word Problem Solving with Operation-Based Formalisms. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2357–2367. https://arxiv.org/abs/1905.13319

Chen, Z., Liu, Z., Lai, P., & Wei, X. (2021). Solving Math Word Problems with Problem Type Classification. *Applied Sciences, 11*(12). https://arxiv.org/abs/2308.13844

Balaji, R., & Ananthanarayanan, N. (2020). SA-MEAS: SymPy-based Automated Mathematical Equations Analysis and Solver. *Procedia Computer Science*, Elsevier, 170, 1121–1128. https://www.sciencedirect.com/science/article/pii/S2352711023002923

Apurwa Yadav, Aarshil Patel, Manan Shah (2021). A Comprehensive Review on Resolving Ambiguities in Natural Language Processing. *Comprehensive Review Paper*. https://www.sciencedirect.com/science/article/pii/S2666651021000127

Reys, B., Lindquist, M. M., Lambdin, D. V., & Smith, N. L. (2017). Student Difficulties in Mathematizing Word Problems in Algebra. *Journal of Mathematical Behavior*, Elsevier, 45, 89–104. http://dx.doi.org/10.12973/eurasia.2016.1299a

Swartz, J. A., Olson, K., & Hard, B. M. (2017). Automatic Classification of Question Difficulty Level: Teachers' Estimation vs. Students' Perception. *Applied Measurement in Education*, 30(3), 211–225. https://ieeexplore.ieee.org/document/6462398

Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi Song, Mingjie Zhan, Hongsheng Li (2023). MathCoder: Seamless Code Integration in LLMs for Enhanced Mathematical Reasoning. *GitHub project documentation*. https://arxiv.org/abs/2310.03731

Wu, C.-Y., & Li, C.-T. (2021). Structure-Unified M-Tree Coding Solver for Math Word Problem. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2345–2356. https://arxiv.org/abs/2210.12432

Multiple Authors. (2024). InternLM-Math: Math Problem Solving via Chain-of-Thought, Code Execution, and Formal Proof Verification. *arXiv preprint arXiv:2402.06332*. https://arxiv.org/abs/2402.06332

Anthony W.F. Lao, Philip I.S. Lei (2023). Subject Classification and Difficulty Ranking of Math Problems. *IEEE Transactions on Education*, 62(3), 192–201. https://ieeexplore.ieee.org/abstract/document/10392820

Hoffmann, J., Borgeaud, S., Mensch, A., & Rutherford, E. (2022). Solving Quantitative Reasoning Problems with Language Models. *NeurIPS 2022 Conference Proceedings*. https://arxiv.org/abs/2206.14858

Multiple Authors. (2023). LILA: A Unified Benchmark for Mathematical Reasoning. *arXiv preprint arXiv:2210.17517*. https://arxiv.org/abs/2210.17517

Wolfram, S. (2009). Wolfram Alpha: A Computational Knowledge Engine. *Wolfram Alpha LLC*. Available at https://www.wolframalpha.com

Meurer, A., Smith, C. P., Paprocki, M., Čertík, O., Kirpichev, S. B., Rocklin, M., Silva, A. M. R., et al. (2017). SymPy: Symbolic Mathematics in Python. *PeerJ Computer Science*, 3, e103. https://doi.org/10.7717/peerj-cs.103

Li, Z., Zhang, W., Yan, C., Zhou, Q., Li, C., Liu, H., & Cao, Y. (2023). Seeking Patterns, Not just Memorizing Procedures: Contrastive Learning for Solving Math Word Problems. https://arxiv.org/abs/2110.08464