Math Solver Agent with Step-by-Step

Explanation

TEAM 11

|  |  |
| --- | --- |
| TEAM LEAD | V.Mohana Krishna(22B21A4332) |
| TEAM MEMBERS |  |
|  | M.Rishanth (22B21A4341) |
|  | G.Akhil Kumar (22B21A4337) |
|  | K.Sesi Kumar (22B21A4372)  V.Kalyani (22B21A4305) |

DEPARTMENT : ARTIFICIAL INTILLIGENCE

INTITUTE : KAKINADA INTITUTE OF

ENGINEERING AND TECHNOLOGY

ACADEMIC YEAR : 2025 - 2026

# ABSTRACT

This project is about creating an AI-based Math Solver Agent that can solve different types of math problems and also explain the steps clearly. The system uses the Python library SymPy to do the actual math operations like solving equations, finding derivatives, or calculating integrals. After solving the problem, an AI language model is used to change the steps into simple, studentfriendly explanations in natural language.

The aim of this project is not only to give the correct answer but also to show how the answer is found. Many existing calculators and apps give only the final solution, which makes it difficult for students to learn. Our system is different because it guides the learner step by step, just like a teacher would.

The expected outcome of the project is a smart, easyto-use, and educational tool that can improve the way students practice mathematics. It will help learners build confidence, understand concepts better, and reduce their dependence on just memorizing formulas.

# Introduction & Problem Statement

Mathematics plays a crucial role in education and problem-solving across multiple fields. However, many students struggle to understand how to approach and solve problems step by step. Existing calculators and tools often provide only final answers, leaving learners without the reasoning process. This gap makes it difficult for students to strengthen their conceptual understanding.

The objective of this project is to build a Math Solver Agent that can automatically solve a variety of mathematical problems and explain the solution process in a structured, easy-to-follow format.

# Why it is important

The project is also important because it can bridge the gap between simple digital calculators and advanced intelligent tutoring systems. While calculators are fast, they do not explain the reasoning. On the other hand, human teachers explain but may not always be available. Our system provides both — speed and explanation — in one platform.

# Proposed Methodology

The proposed system combines symbolic computation with AI-based text generation to solve and explain mathematical problems. The methodology includes:

## 1. Data Gathering

In this step, we collect mathematical problems along with their step-by-step solutions to train and evaluate the Math Solver Agent. Since the project is built from scratch and does not use any external APIs, we rely on publicly available **open-source math datasets** and manually prepared problem sets

* **MATH Dataset (Hendrycks et al.)** – A large dataset with **12,500+ competitionlevel math problems** covering algebra, calculus, probability, geometry, and more.
* **GSM8K (Grade School Math 8K)** – Around **8,500 grade-school level problems**, each with step-by-step reasoning, useful for training the explanation module.
* **ASDiv Dataset** – A collection of **2,300 arithmetic and algebraic word problems** with annotated solutions.

## 2. Data Preprocessing

1. Convert raw dataset items into a **uniform structured format** your training/inference pipeline expects.
2. Normalize math notation so SymPy can parse it.
3. Create aligned pairs: **(symbolic step representation) → (natural-language explanation)**.
4. Produce clean **train / validation / test** splits and quality checks.

## 3. Model Selection (suitable choices from scratch)

## Sequence-to-Sequence (Seq2Seq) Model with Encoder–Decoder architecture

## Encoder: Reads the math step (before → after).

## Decoder: Generates the natural language explanation word by word.

## Training data: Aligned pairs of symbolic steps and human explanations (from datasets like GSM8K, ALG514, MATH).

**Architecture Choices**

1. **RNN-based Seq2Seq (simpler, traditional)**
   * Encoder: LSTM/GRU (reads input step tokens).
   * Decoder: LSTM/GRU (outputs explanation tokens).
   * Can be trained from scratch on a small dataset (thousands of pairs).
2. **Transformer-based Seq2Seq (modern, powerful)**
   * Encoder: Transformer layers (attention-based).
   * Decoder: Transformer layers that attend to encoder outputs.
   * More accurate for complex explanations, but needs more data.

## 4. Parameter Tuning (practical suggestions)

For Template/Retrieval

* No hyperparameters except ranking thresholds (TF-IDF top-k, similarity cutoff).

For Small Transformer (if used)

* Optimizer: **AdamW**
* LR: **5e-4** with linear warmup (1000 steps) and decay.
* Batch size: **32** (GPU) or **8–16** (CPU).
* Epochs: **10–30** (monitor validation loss).
* Label smoothing: 0.1, Dropout: 0.1
* Beam size at inference: 3

## 5. Training

* Prepare tokenized input-output pairs.
* Use teacher forcing in training. Monitor val loss and BLEU/ROUGE for checkpoints.
* Save best checkpoint by val loss.

Monitoring

* Log training/val loss curves.
* Use sample outputs to inspect fluency and correctness.

## 6. Testing / Evaluation Functional tests

* **Final answer accuracy**: compare model final answer with ground truth.
* **Step validity %**: verify each before→after with SymPy simplification (should hold).

Explanation quality

* Automatic: BLEU/ROUGE/BERTScore (for seq2seq).
* Human: collect ratings (1–5) for clarity, correctness, usefulness on random sample (100–200).

Performance

* Average inference time per problem (ms). Memory footprint.

Robustness

* Test on varied notations, unseen problem types, small noisy inputs.

## 7. Save Models & Artifacts

Seq2Seq

* Save PyTorch model weights: model\_best.pt
* Save tokenizer/vocab: vocab.model / tokenizer.json
* Save training logs and config: training\_config.yaml, train.log

## 8. Integration & Deployment

Local Web UI

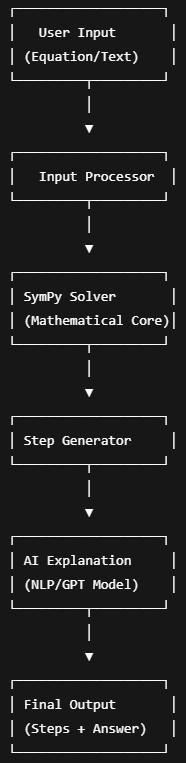
* Build with **Streamlit** (easy) or **Flask**.
* Flow:
  1. User inputs problem text / LaTeX.
  2. Backend parser normalizes and creates SymPy expression.
  3. SymPy solver produces step sequence.
  4. For each step: generate explanation via Template / Retrieval / Model.
  5. Render steps and explanations with LaTeX formatting; offer PDF export (python-docx / ReportLab).

## Tools (must-have)

These are the **foundations** for solving math problems and handling data:

* **Python 3.x** → Main programming language.
* **SymPy** → Symbolic math solving (equations, differentiation, integration, simplification).
* **Pandas** → For handling datasets (CSV/JSON preprocessing).
* **Regex**  → For cleaning and normalizing math notation.
* **JSON / Pickle** → To save templates, processed datasets, and models.

## Flow Diagram System



## Conclusion

The Math Solver Agent with Step-by-Step Explanation is designed to make learning mathematics more interactive and insightful. By not only solving problems but also explaining each step, the system empowers students to grasp fundamental concepts better. The project aims to serve as a bridge between traditional calculators and intelligent tutoring systems, creating a tool that is both practical and educational.

## REFERENCE

1. **MATH Dataset (Hendrycks et al., 2021)**
   * A large dataset with 12,500+ competition-level math problems. Used to train and test AI models on advanced mathematics.
2. **GSM8K Dataset (Cobbe et al., 2021)**
   * A collection of grade-school math problems with step-by-step solutions. Useful for training models to give clear explanations.
3. **ASDiv Dataset (Miao et al., 2020)**
   * Arithmetic and algebraic word problems with annotated solutions. Helps in building reasoning-based systems.