AGENTIC MATHS AI: LLM, RAG & WEB
Comprehensive System Documentation

Developed & Presented by: Pranay Akhil Jeedimalla
jpranayakhil066@gmail.com GitHub LinkedIn
(click here for source code)

Introduction to the Math Agent

The Math Agent represents an intelligent, real-time mathematical problem-solving assistant built using LLMs, knowledge retrieval, and web research capabilities. Designed to mimic the behavior of a diligent and responsible math professor, it engages with users in an informed, context-aware, and transparent manner.

Unlike static solvers, it offers dynamic decision-making: quickly identifying whether a problem is simple enough for direct resolution, suitable for knowledge-based recall, or novel enough to require internet research. The entire flow is implemented in a lightweight but functional Python-FastAPI application, with practical features like feedback loops and explicit user confirmations.

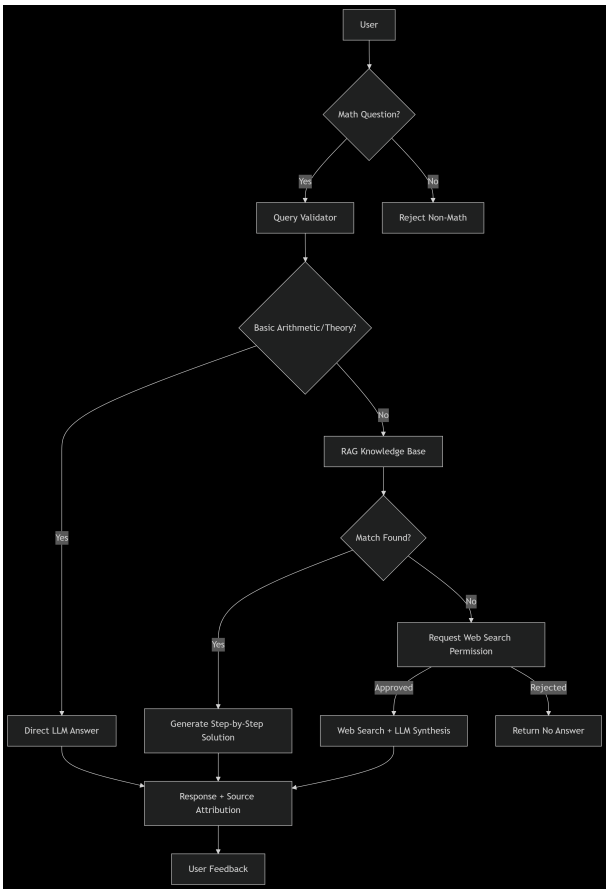

Figure: High-Level Design (HLD) of the System

Architectural Philosophy

At its core, the system embodies three key design principles:

```python
def ask_agent(self, query):
    # acknowledging the user feedback for previous response in history
    if self.check_feedback():
        fb_ack = "\n[User was dissatisfied with a previous response]"
        if self.history and self.history[-1]['role'] == 'assistant':
            self.history[-1]['content'] += fb_ack
    try:
        self.last_query = query
        # query validating
        if not is_math_question(query):
            return "Sorry, I can only answer mathematics-related questions. Please ask a math que

        # answer directly with LLM
        if is_basic_arithmetic_or_theory(query):
            prompt = f"Student Question: {query}\nPlease answer this directly and simply as a mat
            history = self.history + [{"role": "user", "content": prompt}]
            response = self.LLM(history)
            self.history.append({"role": "assistant", "content": response})
            return response + "\n\n Source: Direct LLM"

        # answer with RAG
        rag_response = self.rag.retrieve(query)
        if rag_response:
            verified_prompt = f"""Student Question: {query}\n\nI found a 'Similar Problem' in my
            self.history.append({"role": "user", "content": verified_prompt})
            response = self.LLM(self.history)
            self.history.append({"role": "assistant", "content": response})
            return response + "\n\n Source: Knowledge Base"
        else:
            # human conformation for web_search
            return {"permission_required": True, "message": "I couldn't find this in my knowledge
    except Exception as e:
        return f"Error: {str(e)}"
```

Figure: Agent's query flow & routing to relevant source utilization

1. **Precision Targeting**
   A custom-built query validator ensures that only mathematically relevant questions proceed through the pipeline. It uses regex-based numeric filters along with keyword-based semantic matching across hundreds of math-related terms to detect mathematical intent.

```python
def is_math_question(text):
    if re.search(r'\d+\.?\d*', text):
        return True
    text_lower = text.lower()
    return any(word in text_lower for word in math_keywords)

# LLM
def is_basic_arithmetic_or_theory(query):
    arithmetic_pattern = r'^[\d\s\+\-\=\?\%\*/\(\)\.]+$'
    if re.match(arithmetic_pattern, query.strip()):
        return True
    theory_starts = [
        'what', 'define', 'explain', 'who', 'formula', 'state', 'meaning',
    ]
    q_lower = query.strip().lower()
    return any(q_lower.startswith(start) for start in theory_starts)
```

Figure: Code - Math Query Validation

2. **Progressive Intelligence**
   The agent mimics human reasoning workflows:

   - Solves basic problems instantly using LLMs.
   - Fetches and adapts structured answers from a Firebase-hosted MathQA-based knowledge base when the problem demands it.
   - Falls back to real-time web search using Serper API if the KB does not contain a match, but only after explicit user permission.

3. **Transparent Operations**
   Every response includes source attribution, clearly marking whether it was generated from the LLM, a KB match, or synthesized using web data. This promotes trust and helps the user understand the reasoning path taken.

Detailed Component Breakdown

A. The Gatekeeper: Input Validation System

The validation system screens out non-mathematical inputs early in the pipeline:

- **Regex-Based Numerical Detector**: Searches for digits, decimals, and arithmetic symbols using regex like `(\d+\.?\d*)`.
- **Keyword Matcher**: Compares queries against a curated CSV list of 500+ math keywords spanning algebra, calculus, geometry, statistics, etc.
- **Theory Structure Detection**: Uses phrase patterns like "what is", "explain", "derive", or "define" to identify theoretical questions even if they lack numbers.

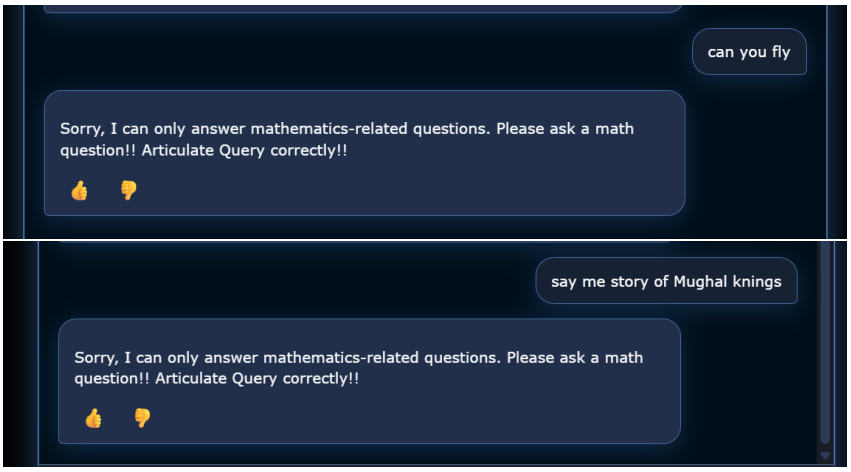

Figure: Input Validation in Action

B. The Decision Engine: Query Routing Logic

Once a query is validated, it is passed through a routing system that chooses one of three answer sources. The design ensures minimal latency for simple problems and comprehensive explanations for complex ones.

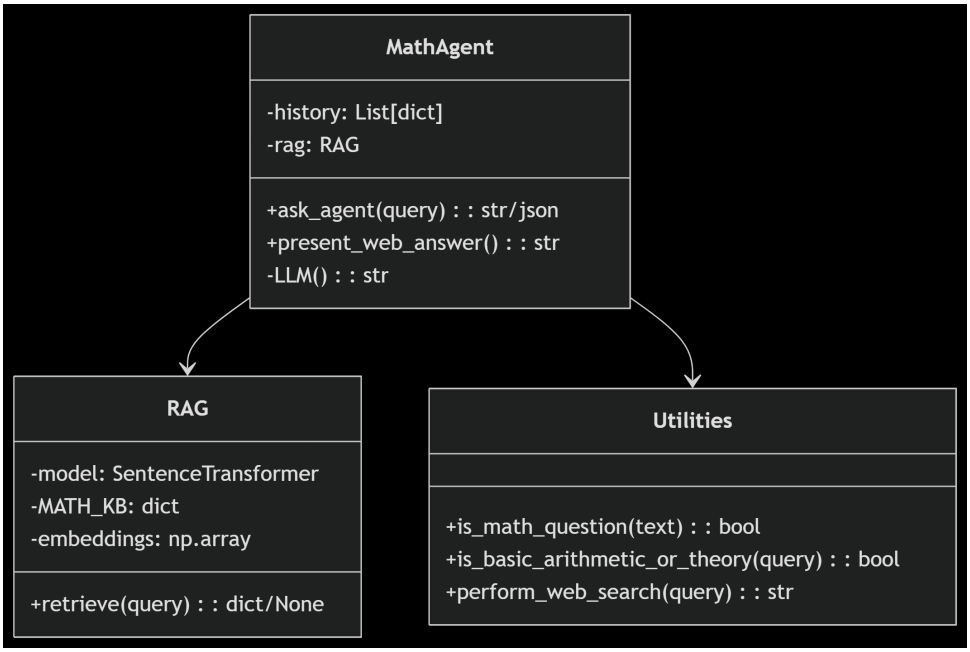

Figure: Low-Level Design (LLD) and Routing System Flow

- **Instant Answer Path (LLM)**

  - Simple calculations ("$2 + 3 \times (4 - 1)$")
  - Basic theoretical definitions ("What is a factorial?")
  - Formula applications with limited steps

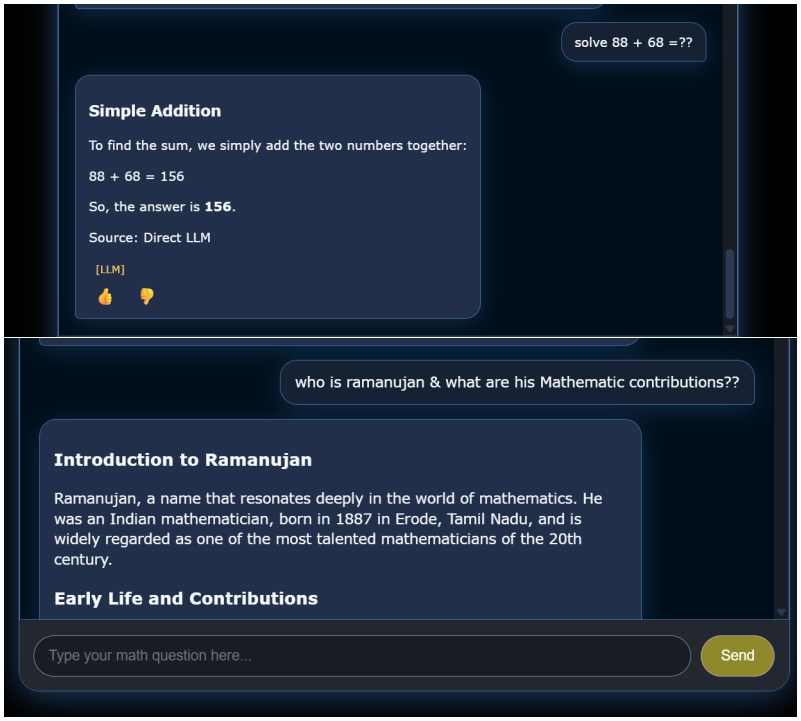

Figure: Direct LLM Response Examples

- **Knowledge Base Path (RAG)**

- Geometry word problems
- Physics-math cross problems
- Exam-style algebra and calculus questions

Figure: KB Response Examples

- **Research Path (Web)**

  - User receives a permission request prompt
  - On approval, Serper is queried
  - Top search result is passed to LLM for synthesis

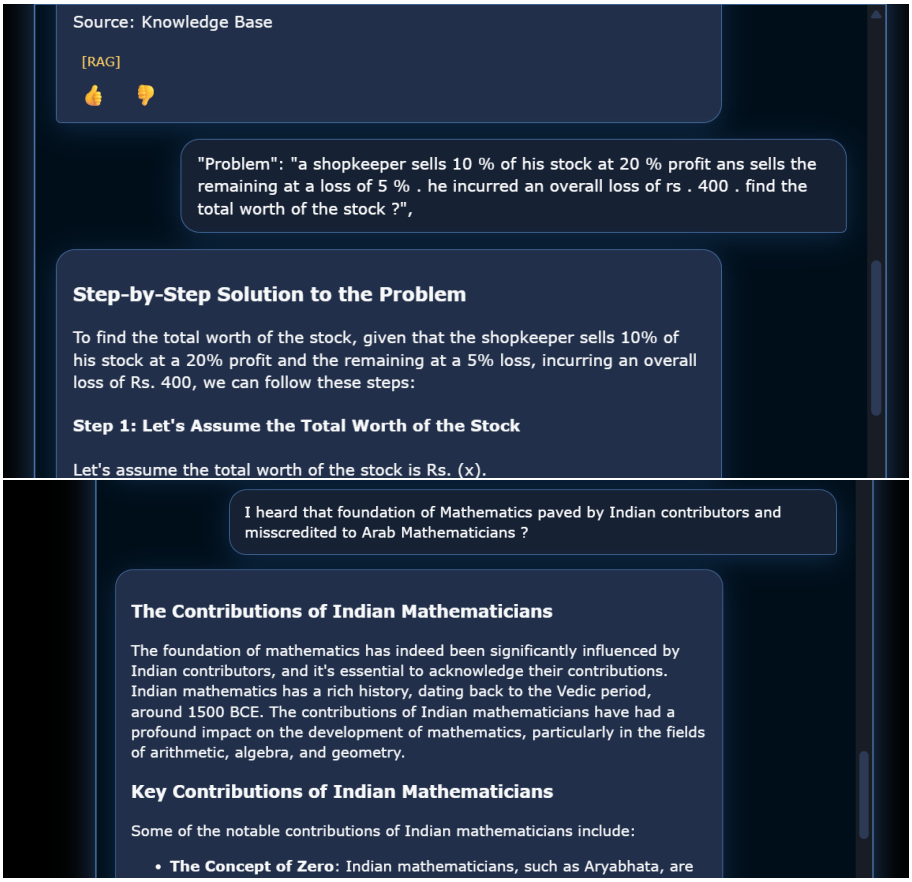

Figure: Web Search-Based Answers

C. The Digital Textbook: Knowledge Base System

The RAG engine uses a curated subset of the MathQA dataset (from hugging face), hosted via Firebase Realtime Database.


```
{
    "Problem": "the speed of a car is 80 km in the first hour and 40 km in the second hou
    "Rationale": "\"s = ( 80 + 40 ) / 2 = 60 kmph answer : b\"",
    "options": "a ) 72 kmph , b ) 60 kmph , c ) 30 kmph , d ) 80 kmph , e ) 82 kmph",
    "correct": "b",
    "annotated_formula": "divide(add(80, 40), const_2)",
    "linear_formula": "add(n0,n1)|divide(#0,const_2)|",
    "category": "physics"
},
{
    "Problem": "two trains 200 m and 250 m long run at the speed of 72 kmph and 18 kmph i
    "Rationale": "\"relative speed = 72 + 18 = 90 kmph * 5 / 18 = 25 m / s distance cover
    "options": "a ) 5.6 sec , b ) 8.9 sec , c ) 10.8 sec , d ) 12.6 sec , e ) 18 sec",
    "correct": "e",
    "annotated_formula": "divide(add(200, 250), multiply(add(72, 18), const_0_2778))",
    "linear_formula": "add(n0,n1)|add(n2,n3)|multiply(#1,const_0_2778)|divide(#0,#2)|",
    "category": "physics"
},
{
    "Problem": "the \u201c length of integer x \u201d refers to the number of prime facto
    "Rationale": "\"to maximize the length of z , we should minimize its prime base . the
    "options": "a ) 7 , b ) 9 , c ) 11 , d ) 13 , e ) 15",
    "correct": "c",
    "annotated_formula": "log(power(2, const_10))",
    "linear_formula": "power(n3,const_10)|log(#0)|",
    "category": "general"
},
```


Figure: MathQA Dataset Sample Structure

Embeddings are computed locally using SentenceTransformer and stored for reuse.

```python
# knowledge base
class rag:
    def __init__(self):
        self.model = SentenceTransformer('all-MiniLM-L6-v2')
        response = requests.get(FIREBASE_KB_URL)
        if response.status_code == 200:
            self.MATH_KB = response.json()
        else:
            raise Exception(f"Failed to load knowledge base: {response.status_code}")

        self.problems = [item['Problem'] for item in self.MATH_KB]
        if os.path.exists('knowledge/math_kb_embeddings.npy'):
            self.embeddings = np.load('knowledge/math_kb_embeddings.npy')
        else:
            self.embeddings = self.model.encode(self.problems)
            np.save('knowledge/math_kb_embeddings.npy', self.embeddings)

    def retrieve(self, query, threshold=0.76):
        query_embedding = self.model.encode([query])
        similarities = cosine_similarity(query_embedding, self.embeddings)[0]
        most_similar = np.argmax(similarities)
        if similarities[most_similar] > threshold:
            return self.MATH_KB[most_similar]
        return None
```

Figure: RAG Engine Implementation Code

D. The Research Assistant: Web Integration

```python
# Web Search
def perform_web_search(query):
    if not SERPER_API_KEY:
        return "Web search is not available (API key missing)."
    url = "https://google.serper.dev/search"
    headers = {"X-API-KEY": SERPER_API_KEY, "Content-Type": "application/json"}
    data = {"q": query, "gl": "in", "hl": "en"}
    try:
        resp = requests.post(url, headers=headers, json=data, timeout=10)
        resp.raise_for_status()
        results = resp.json().get("organic", [])
        if not results:
            return "Sorry, I could not find a reliable answer online."
        snippet = results[0].get("snippet")
        link = results[0].get("link")
        return f"Web Search Result: {snippet}\n(Source: {link})"
    except Exception as e:
        return f"Web search error: {str(e)}"
```

```python
def present_web_answer(self, user_question, web_result):
    prompt = f"""Student Question: {user_question}\n\nHere is a web search
    history = self.history + [{"role": "user", "content": prompt}]
    response = self.LLM(history)
    self.history.append({"role": "assistant", "content": response})
    return response + "\n\n Source: Internet Source"
```

Figure: Web Search Logic Using Serper API

Responses are generated by combining web information with LLM-based synthesis.

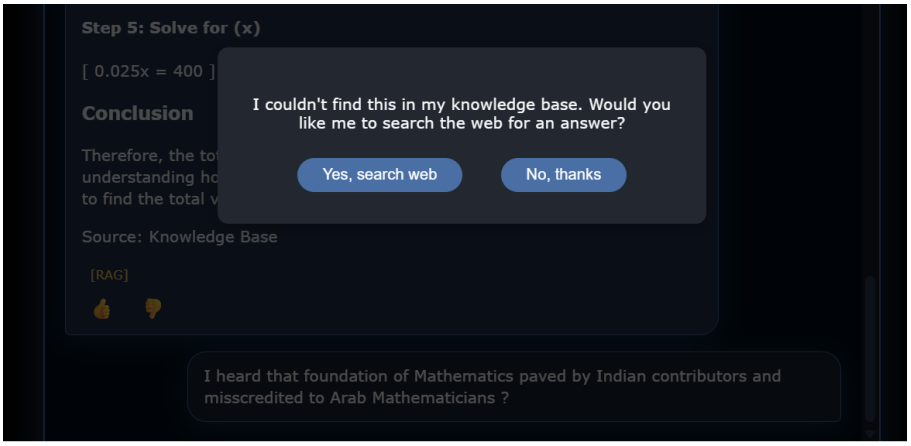

Figure: User Permission Request for Web Search

The Brain: LLM Orchestration

LLM response generation is handled by Groq-hosted LLaMA-3.3-70B with controlled temperature for deterministic and factual answers.

```python
def LLM(self, history):
    try:
        response = client.chat.completions.create(
            messages=history,
            max_tokens=2025,
            model="llama-3.3-70b-versatile",
            temperature=0.3)
        return response.choices[0].message.content
    except Exception as e:
        return f"Error: {str(e)}"
```

Figure: LLM Interaction Code Snippet

Learning Mechanisms

A. Explicit Feedback System

User feedback is stored in a JSON file, linking each question with its answer and vote.


```
},
{
  "question": "I heard that foundation of Mathematics paved by Indian contributors a
  "answer": "### Indian Contributions to Mathematics\nThe history of mathematics is
  "feedback": "up"
},
{
  "question": "who is karl marx and what is his philosophy of life??",
  "answer": "### Response\nAs a math professor, I must politely decline to answer qu
  "feedback": "up"
},
{
  "question": "hi",
  "answer": "Sorry, I can only answer mathematics-related questions. Please ask a ma
  "feedback": "up"
}
```

Figure: Feedback Storage Format

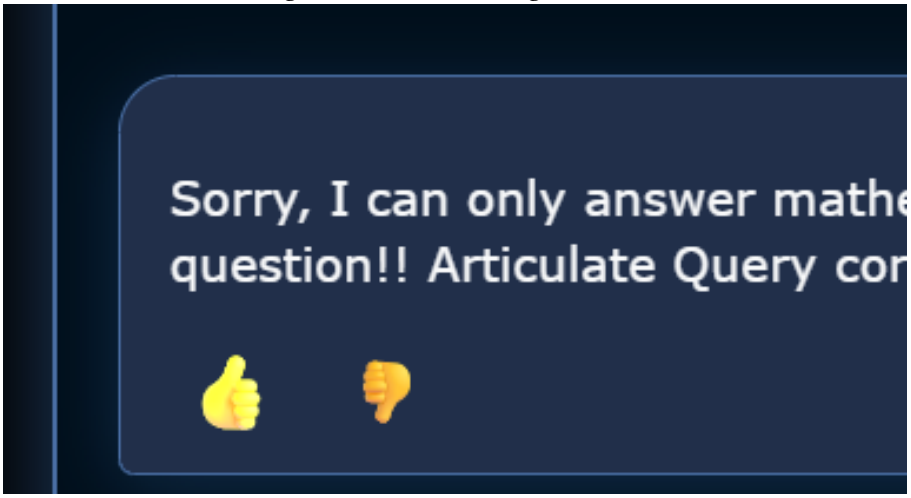

Figure: Feedback Interface (Thumbs Up/Down)

B. Implicit Optimization

- Local caching of embeddings
- Feedback referencing for future refinement
- Logging of unanswered queries for knowledge enrichment

Operational Characteristics

Performance Considerations

- Fast cosine similarity search
- Real-time updates from Firebase
- System fallback mechanisms during external failures

Error Management

- Comprehensive exception handling
- Error logs and safe fallbacks

Visual Clarity and Transparency

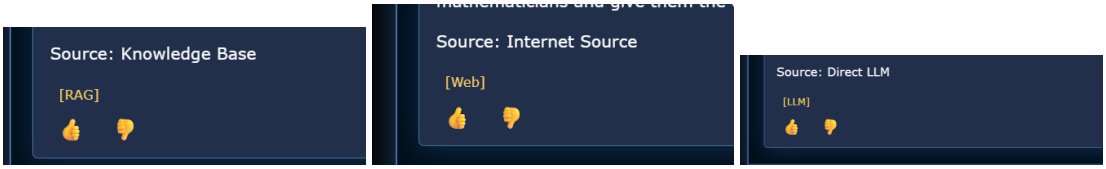

Figure: Source Attribution on UI (KB, Web, LLM)

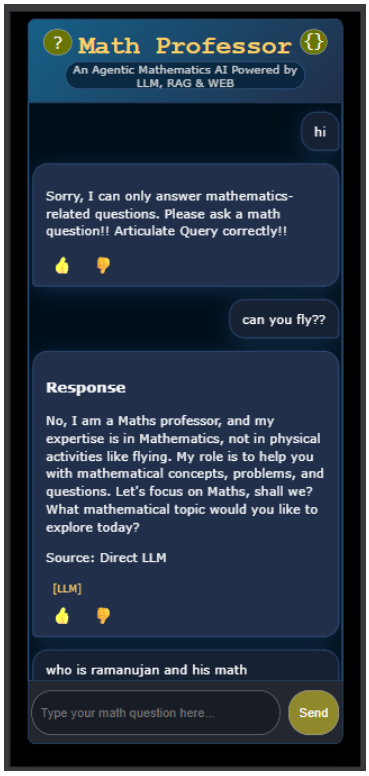

Figure: Agent UI - Mobile

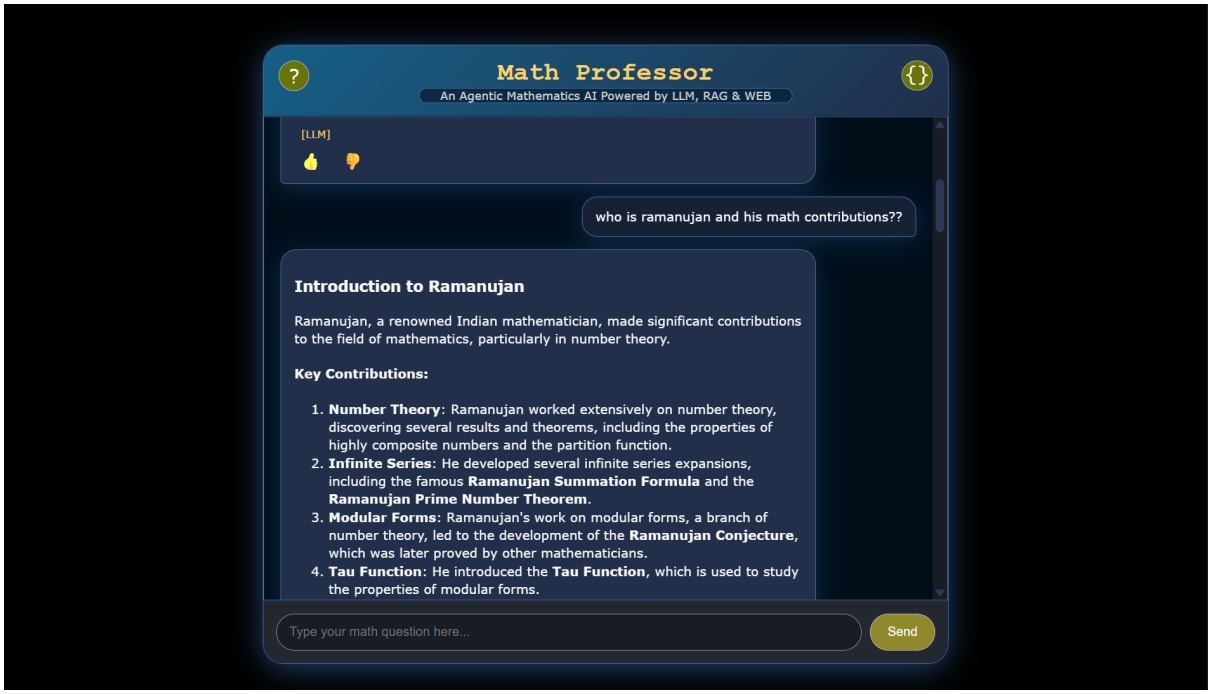

Figure: Agent UI - Web

Future Development Pathways

Short-Term Enhancements

- Adaptive thresholds
- Multi-turn explanation support
- Symbolic math parsing

Long-Term Vision

- Deploy this Agent as a website by implementing better approaches to optimize the issues and make it available on-line.
- Visual proofing and math drawings
- Personalization via user history

Conclusion

Math-Agent is a working example of practical AI built with real-time intelligence and human-centric design. By intelligently routing queries through arithmetic logic, structured knowledge, and adaptive search, it offers both speed and pedagogical depth. Its modular construction ensures extendability and maintenance, while the respect for user control and transparency sets a benchmark for AI in education.

note: I tried to deploy this agent as a website on vercel, railways, render, etc. But, failed because of high data & memory consumption issues (like "sentence-transformers", "py libraries", etc). I am now constantly trying different approaches & frameworks to optimize the issues and deploy it on-line.

Project Source code

- The complete source code for this project is available at: `https://github.com/akhilbrucelee066/MathAgent-codebase/`

Reference

- MathQA dataset : https://huggingface.co/datasets/allenai/math_qa