Math Mentor AI - Source Code Documentation

Document Information

Field	Value
Project	Math Mentor AI
Version	1.0
Date	January 2026
Repository	https://github.com/Ashwadhama2004/mL-project

---

1. Project Structure Overview

```
math-mentor-ai/
│
├── app.py                            # Main Streamlit application entry
├── requirements.txt                  # Python dependencies (31 packages)
├── packages.txt                      # System dependencies for cloud
├── .env.example                      # Environment template
│
├── agents/                           # Multi-Agent System (5 agents)
│   ├── __init__.py
│   ├── parser_agent.py              # Input parsing & topic detection
│   ├── router_agent.py              # Strategy routing & decision
│   ├── solver_agent.py              # Core solution generation
│   ├── verifier_agent.py            # Solution validation
│   └── explainer_agent.py           # Pedagogical explanation
│
├── input_processors/                 # Multimodal Input Handling
│   ├── __init__.py
│   ├── text.py                      # Text validation
│   ├── ocr.py                       # Image OCR with LLM enhancement
│   └── asr.py                       # Audio transcription with Gemini
│
├── rag/                             # RAG System
│   ├── __init__.py
│   ├── build_index.py               # FAISS index builder
│   ├── retriever.py                 # Semantic retrieval interface
│   └── knowledge_base/              # 17 markdown knowledge docs
│       ├── algebra.md
│       ├── calculus.md
│       ├── trigonometry.md
│       └── ... (14 more)
│
├── memory/                          # Self-Learning System
│   ├── __init__.py
│   └── memory_store.py              # SQLite operations
│
```

```
│
├── utils/                          # Shared Utilities
│   ├── __init__.py
│   ├── llm_client.py               # Gemini API wrapper
│   ├── tools.py                    # Python calculator
│   ├── confidence.py               # Scoring utilities
│   └── logger.py                   # Structured logging
│
├── data/                           # Persistent Storage
│   ├── faiss_index/                # Vector embeddings
│   │   ├── index.faiss
│   │   ├── chunks.pkl
│   │   └── metadata.json
│   └── memory_store.db             # SQLite database
│
├── docs/                           # Documentation
│   ├── app_homepage.png
│   ├── app_solution.png
│   └── demo.webp
│
└── .streamlit/                     # Streamlit config
    └── config.toml
```

---

2. Core Components

2.1 app.py - Main Application

Location: `app.py` (629 lines)

Purpose: Streamlit web application entry point that orchestrates all components.

Key Functions:

Function	Lines	Description
`init_session_state()`	66-81	Initialize Streamlit session variables
`render_sidebar()`	84-135	Render settings panel with sliders
`render_input_section()`	138-186	Handle multimodal input (text/image/audio)
`process_input()`	189-222	Route input to appropriate processor
`run_agent_pipeline()`	225-340	Execute 5-agent pipeline
`render_solution()`	420-470	Display solution with formatting
`render_agent_trace()`	380-418	Show agent execution trace
`main()`	521-629	Application entry point

Key Code Pattern:

```
def run_agent_pipeline(problem_text, source, settings):
    """Execute the multi-agent pipeline."""
    trace = []
```

```
    # 1. Parser Agent
    parser = ParserAgent(llm)
    parsed = parser.parse(problem_text)
    trace.append({"agent": "Parser", "status": "completed", ...})

    # 2. Router Agent
    router = RouterAgent(llm)
    route = router.route(parsed)

    # 3. Solver Agent (with RAG)
    solver = SolverAgent(llm, retriever, memory)
    solution = solver.solve(problem_text, route)

    # 4. Verifier Agent
    verifier = VerifierAgent(llm)
    verification = verifier.verify(solution)

    # 5. Explainer Agent
    explainer = ExplainerAgent(llm)
    explanation = explainer.explain(solution)

    return {"success": True, "trace": trace, ...}
```

2.2 Agents Module

parser_agent.py

Location: `agents/parser_agent.py` (180 lines)

Class: `ParserAgent`

Purpose: Parse and structure raw problem input.

Key Methods:

Method	Description
`parse(text: str)`	Main parsing entry point
`_detect_topic(text)`	Identify math topic (algebra, calculus, etc.)
`_extract_variables(text)`	Find variables in expression
`_check_ambiguity(parsed)`	Detect if clarification needed

Output Schema:

```
{
    "problem_text": str,       # Cleaned input
    "detected_topic": str,     # "algebra", "calculus", etc.
    "variables": List[str],    # ["x", "y"]
    "constraints": List[str], # Any constraints
    "needs_clarification": bool,
    "clarification_question": Optional[str]
}
```

---

router_agent.py

Location: `agents/router_agent.py` (150 lines)

Class: `RouterAgent`

Purpose: Decide solving strategy based on parsed problem.

Key Methods:

Method	Description
`route(parsed: dict)`	Generate routing decision
`_map_topic_to_solver(topic)`	Match topic to solver type

Output Schema:

```
{
    "solver_type": str,       # "algebraic_solver", "calculus_solver"
    "use_rag": bool,          # Should query knowledge base
    "use_calculator": bool,   # Should use Python calc
    "difficulty": str,        # "basic", "intermediate", "advanced"
    "rag_filters": List[str]  # Topics to filter RAG results
}
```

---

solver_agent.py

Location: `agents/solver_agent.py` (350 lines)

Class: `SolverAgent`

Purpose: Core solution generation with RAG integration.

Key Methods:

Method	Description
`solve(problem, route_config)`	Main solving method
`_get_rag_context(query, k)`	Retrieve knowledge chunks
`_check_memory(problem)`	Look for similar solved problems
`_build_prompt(problem, context)`	Construct LLM prompt
`_calculate_confidence(solution)`	Compute confidence score

RAG Integration:

```
def _get_rag_context(self, query, k=5):
    """Retrieve relevant knowledge from FAISS index."""
    chunks = self.retriever.retrieve(query, k=k)
    context = "\n\n".join([
        f"[Source: {c['source']}]\n{c['content']}"
        for c in chunks
    ])
```

```
    return context, chunks
```

Confidence Calculation:

```
confidence_factors = {
    "rag_coverage": 0.9 if has_context else 0.3,
    "citation_quality": 0.9 if has_citations else 0.4,
    "llm_confidence": float(solution.get("confidence", 0.7)),
    "has_verification": 0.9 if verified else 0.6
}
final_confidence = sum(confidence_factors.values()) / len(confidence_fa
```

---

verifier_agent.py

Location: `agents/verifier_agent.py` (180 lines)

Class: `VerifierAgent`

Purpose: Validate solution correctness and calculate confidence.

Key Methods:

Method	Description
`verify(solution: dict)`	Main verification method
`_check_logical_consistency(steps)`	Validate reasoning
`_verify_arithmetic(solution)`	Use Python to check math
`_calculate_final_score(factors)`	Weighted confidence

HITL Trigger Logic:

```
if final_confidence < self.hitl_threshold:  # 0.70
    return {
        "verdict": "uncertain",
        "hitl_required": True,
        "question": "Please verify this solution..."
    }
```

---

explainer_agent.py

Location: `agents/explainer_agent.py` (200 lines)

Class: `ExplainerAgent`

Purpose: Generate student-friendly explanations.

Key Methods:

Method	Description
`explain(solution: dict)`	Create pedagogical explanation
`_format_steps(steps)`	Number and format steps

`_add_concepts(topic)`	Include key concepts
`_add_exam_tips(topic)`	Include JEE tips

Output Schema:

```
{
    "final_explanation": str,   # Formatted solution
    "steps": List[str],         # Step-by-step breakdown
    "key_concepts": List[str],  # Concepts used
    "exam_tips": List[str]      # JEE exam tips
}
```

2.3 Input Processors Module

ocr.py

Location: `input_processors/ocr.py` (350 lines)

Class: `OCRProcessor`

Purpose: Extract text from math problem images with LLM enhancement.

Key Methods:

Method	Description
`process(image_input)`	Main OCR processing
`process_bytes(image_bytes)`	Process from bytes
`_normalize_text(text)`	Fix common OCR errors
`_llm_enhance_ocr(text, conf)`	Use Gemini to fix symbols

LLM Enhancement (Lines 123-167):

```
def _llm_enhance_ocr(self, raw_text: str, confidence: float) -> str:
    """Use LLM to fix OCR-extracted math text."""
    prompt = f"""You are a math OCR correction expert...
    Common OCR misreadings:
    - Division symbol (÷) often misread as "-:", ":", "-"
    - Multiplication (×) misread as "x" or "*"

    Original OCR text: "{raw_text}"

    Output ONLY the corrected expression:"""

    corrected = self.llm.generate(prompt)
    return corrected if corrected else raw_text
```

---

asr.py

Location: `input_processors/asr.py` (454 lines)

Class: `ASRProcessor`

Purpose: Transcribe audio of spoken math problems.

Key Methods:

Method	Description
`process_bytes(audio_bytes)`	Process audio bytes
`_transcribe_with_gemini(bytes)`	Gemini audio transcription
`_normalize_math_phrases(text)`	Convert speech to symbols

Gemini Audio Transcription (Lines 145-238):

```
def _transcribe_with_gemini(self, audio_bytes: bytes):
    # Detect audio format
    mime_type = "audio/webm"  # Default for st.audio_input
    if audio_bytes[:4] == b'RIFF':
        mime_type = "audio/wav"

    # Create inline data for Gemini
    audio_part = {
        "inline_data": {
            "mime_type": mime_type,
            "data": base64.b64encode(audio_bytes).decode('utf-8')
        }
    }

    # Transcribe with Gemini
    model = genai.GenerativeModel("gemini-2.0-flash")
    response = model.generate_content([prompt, audio_part])
    return response.text
```

Math Phrase Normalization:

```
MATH_PHRASES = {
    r'\bplus\b': '+',
    r'\bminus\b': '-',
    r'\btimes\b': '×',
    r'\bdivided by\b': '÷',
    r'\bsquared\b': '²',
    r'\bcubed\b': '³',
    ...
}
```

2.4 RAG Module

retriever.py

Location: `rag/retriever.py` (270 lines)

Class: `RAGRetriever`

Purpose: Semantic search over knowledge base.

Key Methods:

Method	Description
`__init__()`	Load FAISS index and chunks
`retrieve(query, k=5)`	Main retrieval method
`_embed_query(text)`	Generate query embedding

Retrieval Flow:

```
def retrieve(self, query: str, k: int = 5) -> List[Dict]:
    """Retrieve top-k relevant chunks."""
    # 1. Embed query
    query_vector = self.model.encode([query])

    # 2. Search FAISS index
    scores, indices = self.index.search(query_vector, k)

    # 3. Filter by threshold
    results = []
    for score, idx in zip(scores[0], indices[0]):
        if score >= self.threshold:  # 0.5
            results.append({
                "content": self.chunks[idx]["text"],
                "source": self.chunks[idx]["source"],
                "score": float(score)
            })

    return results
```

build_index.py

Location: `rag/build_index.py` (180 lines)

Purpose: Build FAISS index from knowledge base.

Process:

1. Load all .md files from knowledge_base/
2. Chunk text (~500 chars with 100 overlap)
3. Generate embeddings with all-MiniLM-L6-v2
4. Build FAISS IndexFlatIP
5. Save index.faiss, chunks.pkl, metadata.json

2.5 Memory Module

memory_store.py

Location: `memory/memory_store.py` (200 lines)

Class: `MemoryStore`

Purpose: SQLite-based problem memory and feedback.

Key Methods:

Method	Description
`save_problem(problem, solution)`	Store solved problem
`find_similar(query, threshold)`	Find similar problems
`update_feedback(id, feedback)`	Record user feedback
`get_stats()`	Return memory statistics

Database Schema:

```
CREATE TABLE problems (
    id TEXT PRIMARY KEY,
    problem_text TEXT,
    problem_hash TEXT,
    solution TEXT,
    confidence REAL,
    feedback TEXT,
    created_at TIMESTAMP,
    source TEXT
);

CREATE TABLE feedback_log (
    id INTEGER PRIMARY KEY,
    problem_id TEXT,
    feedback_type TEXT,
    user_correction TEXT,
    timestamp TIMESTAMP
);
```

2.6 Utils Module

llm_client.py

Location: `utils/llm_client.py` (275 lines)

Class: `LLMClient`

Purpose: Unified Gemini API interface.

Key Methods:

Method	Description
`generate(prompt, system)`	Text generation
`generate_json(prompt, schema)`	JSON structured output
`_parse_json_response(text)`	Extract JSON from response

API Key Resolution:

```
def __init__(self):
```

```
    # 1. Direct parameter
    # 2. Environment variable
    # 3. Streamlit secrets
    self.api_key = api_key or os.getenv("GOOGLE_API_KEY")

    if not self.api_key:
        import streamlit as st
        if hasattr(st, 'secrets') and 'api_keys' in st.secrets:
            self.api_key = st.secrets.api_keys.get("GOOGLE_API_KEY")
```

---

3. Key Interactions

3.1 Request Flow

```
User Input
    │
    ▼
app.py::render_input_section()
    │ (Text/Image/Audio)
    ▼
input_processors/[text|ocr|asr].py
    │ (Extracted text + confidence)
    ▼
app.py::run_agent_pipeline()
    │
    ├──▶ agents/parser_agent.py
    │           │
    │           ▼
    ├──▶ agents/router_agent.py
    │           │
    │           ▼
    ├──▶ agents/solver_agent.py ◀──┬── rag/retriever.py
    │           │                  └── memory/memory_store.py
    │           ▼
    ├──▶ agents/verifier_agent.py
    │           │
    │           ▼
    └──▶ agents/explainer_agent.py
                │
                ▼
        Result Dictionary
                │
                ▼
app.py::render_solution()
```

3.2 RAG Integration

```
SolverAgent.solve()
    │
    ├── Check memory for similar problems
    │      └── memory_store.find_similar()
    │
```

```
    │
    ├── Query RAG for knowledge
    │      └── retriever.retrieve(query, k=5)
    │             ├── Embed query
    │             ├── FAISS search
    │             └── Return top chunks
    │
    ├── Build context-enriched prompt
    │      └── Combine: problem + RAG chunks + memory hints
    │
    └── Generate solution with LLM
           └── llm_client.generate_json()
```

---

4. Configuration

4.1 Environment Variables

Variable	Description	Default
`GOOGLE_API_KEY`	Gemini API key	Required
`MODEL_NAME`	LLM model	gemini-2.0-flash

4.2 Configurable Parameters

Parameter	Location	Default
OCR Confidence Threshold	Sidebar slider	0.75
Verifier Confidence Threshold	Sidebar slider	0.70
RAG Top-K Results	Sidebar slider	5
LLM Temperature	llm_client.py	0.7
Chunk Size	build_index.py	500 chars
Embedding Model	retriever.py	all-MiniLM-L6-v2

---

5. Error Handling

5.1 LLM Errors

```
# solver_agent.py
try:
    solution = self.llm.generate_json(prompt)
except Exception as json_error:
    # Fallback to text generation
    text_response = self.llm.generate(prompt)
    solution = self._parse_text_response(text_response)
```

5.2 Type Coercion

```
# Fix "unhashable type: list" error
llm_conf = solution.get("confidence", 0.7)
```

```
if isinstance(llm_conf, (list, tuple)):
    llm_conf = float(llm_conf[0]) if llm_conf else 0.7
```

5.3 RAG Fallback

```
# If RAG retrieval fails
try:
    chunks = retriever.retrieve(query)
except Exception:
    chunks = []  # Proceed without RAG
```

---

6. Testing

6.1 Unit Tests

```
# Run all tests
python -m pytest tests/

# Test specific module
python -m pytest tests/test_solver.py -v
```

6.2 Manual Testing

```
# Test RAG retriever
python -m rag.retriever

# Test OCR processor
python -m input_processors.ocr

# Test ASR processor
python -m input_processors.asr
```

---

7. Deployment

7.1 Local Development

```
# Install dependencies
pip install -r requirements.txt

# Build RAG index
python -m rag.build_index

# Run app
streamlit run app.py
```

7.2 Streamlit Cloud

1. Push to GitHub
2. Connect on share.streamlit.io

3. Set secrets: `GOOGLE_API_KEY`
4. Deploy

Document Version: 1.0
Last Updated: January 2026