📚 MAKAI: Makerere AI Assistant - Comprehensive Documentation

🎯 Project Overview

MAKAI (Makerere AI Assistant) is a locally-running AI-powered chatbot that provides information about Makerere University. It combines **Ollama** for local LLM inference, **Qdrant** for vector search, and **PHP** for the web interface, all running locally without external APIs.

Core Architecture

text

```
User Interface (Browser)
    ↓
 PHP Frontend
    ↓
 PHP Backend API
    ↓
Qdrant (Vector DB)
    ↓
Ollama (Embeddings + LLM)
```

📁 Project Structure

text

```
makai-demo/
├── 📁 Frontend
│   ├── index.php          # Main landing page with chat widget
│   ├── chat.php           # Full-screen chat interface
│   ├── styles/output.css  # Compiled Tailwind CSS
│   └── src/input.css      # Tailwind source
│
├── 📁 Backend
│   ├── ask.php            # Main question handler
│   ├── config.php         # Configuration & session management
```

```
│   ├── embed.php          # Text embedding functions
│   ├── qdrant.php         # Vector search operations
│   ├── generate.php       # AI response generation
│   └── ingest.php         # Document ingestion pipeline
│
├── 📁 Data
│   └── knowledge/         # Knowledge base text files
│       ├── aims.txt
│       ├── muele.txt
│       └── fees.txt
│
├── 📁 Configuration
│   ├── package.json       # Node.js dependencies
│   └── tailwind.config.js # Tailwind configuration
│
└── 📁 Infrastructure
    ├── Docker (Qdrant)
    └── Ollama CLI
```

🔧 Technology Stack

Component	Technology	Purpose	Version/Model
LLM/Embeddings	Ollama	Local model inference	nomic-embed-text, llama3.2
Vector Database	Qdrant	Vector storage & search	Docker container
Backend	Plain PHP	Web server & logic	PHP 8.0+
Frontend	Tailwind CSS	Styling	3.4+

Component	Technology	Purpose	Version/Model
JavaScript	Vanilla JS	UI interactions	ES6+
Server	Built-in PHP	Local development	-

🚀 Setup & Installation

Prerequisites Checklist

- PHP 8.0+ installed
- Composer (optional)
- Node.js & npm
- Docker Desktop
- Ollama installed
- 4GB+ RAM available

Step-by-Step Installation

1. Environment Setup

bash

```
# Clone or create project directory
mkdir makai-demo
cd makai-demo

# Create folder structure
mkdir -p knowledge styles assets src
```

2. Install Ollama

bash

```
# On macOS/Linux
curl -fsSL https://ollama.ai/install.sh | sh

# On Windows (PowerShell)
```

```
winget install Ollama.Ollama

# Pull required models
ollama pull nomic-embed-text    # 768-dimensional embeddings
ollama pull llama3.2          # Or mistral, qwen2.5
```

3. Setup Qdrant with Docker

```
bash
# Pull and run Qdrant
docker run -p 6333:6333 -p 6334:6334 \
   -v $(pwd)/qdrant_storage:/qdrant/storage \
   qdrant/qdrant

# Verify Qdrant is running
curl http://localhost:6333
```

4. Install Frontend Dependencies

```
bash
# Initialize npm
npm init -y

# Install Tailwind CSS
npm install -D tailwindcss @tailwindcss/forms

# Initialize Tailwind
npx tailwindcss init

# Build CSS
npm run build
```

5. Prepare Knowledge Base

```
bash
```

```
# Add your university documents
echo "Makerere University Admission Requirements..." > knowledge/admission.txt
echo "Tuition Fees Structure..." > knowledge/fees.txt
# Add more .txt files as needed
```

6. Run the System

```
bash
# Terminal 1: Qdrant (already running in Docker)

# Terminal 2: Start PHP server
php -S localhost:8000

# Terminal 3: Monitor logs (optional)
tail -f /path/to/error_log

# Open in browser
open http://localhost:8000
```

⚙️ Core Components Detailed

1. Embedding System (embed.php)

Purpose: Convert text to vector embeddings using Ollama's nomic-embed-text model.

Key Features:

- Handles text truncation for model limits
- Error handling for Ollama connection issues
- Returns 768-dimensional vectors

Implementation Details:

```
php
function embedText($text) {
    // Limit text length (nomic-embed-text has context limit)
    $text = substr(trim($text), 0, 8192);
```

```
    // JSON payload for Ollama API
    $payload = [
        "model" => "nomic-embed-text",
        "prompt" => $text,
        "options" => ["temperature" => 0]
    ];

    // Make HTTP request
    $context = stream_context_create([
        "http" => [
            "method" => "POST",
            "header" => "Content-Type: application/json",
            "content" => json_encode($payload),
            "timeout" => 30  // 30-second timeout
        ]
    ]);

    // Parse response
    $response = @file_get_contents(
        "http://localhost:11434/api/embeddings",
        false,
        $context
    );
}
```

Challenges Solved:

- **Timeout Handling:** Added 30-second timeout for large documents
- **Memory Management:** Truncates text to prevent OOM errors
- **Error Recovery:** Graceful degradation when Ollama is unavailable

2. Vector Database Manager (qdrant.php)

Purpose: Interface with Qdrant for vector storage and similarity search.

Key Features:

- Collection management (create/check)
- Vector similarity search with scoring
- Batch point insertion
- Payload management (source metadata)

Collection Structure:

```
json
{
  "collection_name": "makai",
  "vectors_config": {
    "size": 768,
    "distance": "Cosine"
  },
  "shard_number": 1,
  "replication_factor": 1
}
```

Search Parameters:

```
php
$searchParams = [
    "vector" => $embedding_vector,
    "limit" => 5,                // Top 5 results
    "with_payload" => true,      // Include source text
    "score_threshold" => 0.5,    // Minimum similarity score
    "with_vectors" => false
];
```

Challenges Solved:

- **Connection Stability:** Retry logic for Docker container startup

- **Scoring Threshold**: Filters out irrelevant results (score < 0.5)
- **Metadata Preservation**: Stores source file and chunk index

3. Document Ingestion Pipeline (ingest.php)

Purpose: Process and chunk text documents into Qdrant.

Processing Steps:

1. **File Discovery**: Scan knowledge/ directory for .txt files
2. **Smart Chunking**: Split by paragraphs/sentences (max 500 chars)
3. **Embedding Generation**: Convert chunks to vectors
4. **Batch Insertion**: Store in Qdrant with metadata

Chunking Algorithm:

```
php
function chunkText($text, $maxLength = 500) {
  // Split by paragraphs first
  $paragraphs = explode("\n\n", $text);
  $chunks = [];

  foreach ($paragraphs as $paragraph) {
    // If paragraph is too long, split by sentences
    if (strlen($paragraph) > $maxLength) {
      $sentences = preg_split('/(?<=[.!?])\s+/', $paragraph);
      $currentChunk = '';

      foreach ($sentences as $sentence) {
        if (strlen($currentChunk) + strlen($sentence) <= $maxLength) {
          $currentChunk .= $sentence . ' ';
        } else {
          if (!empty(trim($currentChunk))) {
            $chunks[] = trim($currentChunk);
          }
```

```
            $currentChunk = $sentence . ' ';
        }
      }

      if (!empty(trim($currentChunk))) {
        $chunks[] = trim($currentChunk);
      }
    } else {
      $chunks[] = trim($paragraph);
    }
  }

  return array_filter($chunks, function($chunk) {
    return strlen($chunk) > 50; // Remove tiny chunks
  });
}
```

Challenges Solved:

- **Document Size Variability:** Handles both short and long documents
- **Semantic Chunking:** Keeps related sentences together
- **Duplicate Prevention:** Uses MD5 hashing for chunk IDs
- **Progress Feedback:** Console output for large ingestions

4. AI Response Generator (generate.php)

Purpose: Use Ollama LLM to generate natural language responses from retrieved context.

Prompt Engineering:

```
php
$systemPrompt = "You are MAKAI (Makerere AI Assistant)...
Rules:
1. Answer ONLY from provided context
2. If unsure, say 'I don't have information about that'
```

3. Be concise and factual

4. Cite sources when available

Context:

{$formattedContext}

Question: {$question}";

Context Formatting:

- Includes top 5 most relevant chunks
- Shows similarity scores
- Preserves source metadata
- Limits total context length to prevent token overflow

Challenges Solved:

- **Context Window Management**: Truncates to fit model limits
- **Hallucination Prevention**: Strict "only from context" prompting
- **Source Attribution**: Includes file names in responses
- **Performance**: Caches frequent queries

5. Web Interface (index.php & chat.php)

Purpose: User-friendly interface for interacting with MAKAI.

Key UI Components:

Floating Chat Widget:

- Bottom-right positioned button
- Slide-up animation
- Typing indicators
- Markdown rendering

Features:

- **Real-time Updates**: WebSocket-like feel with JavaScript polling
- **Responsive Design**: Mobile-first Tailwind CSS
- **Accessibility**: ARIA labels, keyboard navigation

- **Session Management**: PHP sessions for chat history

JavaScript Architecture:

```
javascript
class ChatManager {
  constructor() {
    this.history = [];
    this.isLoading = false;
  }

  async sendMessage(message) {
    // Show typing indicator
    this.showTyping();

    // Make API call
    const response = await fetch('/ask.php', {
      method: 'POST',
      body: JSON.stringify({question: message})
    });

    // Parse and display response
    const data = await response.json();
    this.displayResponse(data);

    // Update chat history
    this.history.push({user: message, ai: data.answer});
  }

  displayResponse(data) {
    // Render markdown
```

```
        const html = markdownit().render(data.answer);

        // Add sources if available
        if (data.sources.length > 0) {
            html += this.renderSources(data.sources);
        }

        // Update DOM
        document.getElementById('chat').innerHTML += html;
    }
}
```

🔄 Data Flow

Ingestion Flow

text

```
Text Document
    ↓
Read & Clean
    ↓
Chunk (500 chars)
    ↓
Embed (Ollama)
    ↓
Store (Qdrant)
    ↓
Index Ready
```

Query Flow

text

```
User Question
```

↓

Embed Question

↓

Vector Search (Qdrant)

↓

Retrieve Top 5 Chunks

↓

Format Context

↓

LLM Generation (Ollama)

↓

Format Response

↓

Display to User

🛠️ **Configuration Variables**

config.php - Core Settings

```
php
// Ollama Configuration
define('OLLAMA_HOST', 'http://localhost:11434');
define('EMBEDDING_MODEL', 'nomic-embed-text');  // 768-dim
define('LLM_MODEL', 'llama3.2');              // Response model

// Qdrant Configuration
define('QDRANT_HOST', 'http://localhost:6333');
define('COLLECTION_NAME', 'makai');
define('VECTOR_SIZE', 768);                   // Must match embedding model
define('DISTANCE_METRIC', 'Cosine');          // Cosine similarity
```

```
// Application Settings
define('MAX_CONTEXT_CHUNKS', 5);            // Chunks per query
define('SIMILARITY_THRESHOLD', 0.5);        // Minimum score
define('CHUNK_SIZE', 500);                  // Characters per chunk
define('SESSION_TIMEOUT', 3600);            // 1 hour
```

Environment Variables (Optional)

```
bash
# .env file (if using PHP dotenv)
OLLAMA_HOST=http://localhost:11434
QDRANT_HOST=http://localhost:6333
EMBEDDING_MODEL=nomic-embed-text
LLM_MODEL=llama3.2
DEBUG_MODE=false
```

⚡ Performance Optimizations

1. Caching Strategy

```
php
// Simple file-based caching
function getCachedEmbedding($text) {
    $hash = md5($text);
    $cacheFile = "cache/embeddings/{$hash}.json";

    if (file_exists($cacheFile) &&
        (time() - filemtime($cacheFile)) < 3600) {
        return json_decode(file_get_contents($cacheFile), true);
    }

    return null; // Cache miss
}
```

2. Batch Processing

- Ingest documents in batches of 10 chunks
- Parallel embedding requests (if supported)
- Bulk insert to Qdrant

3. Memory Management

- Stream large files line by line
- Unset variables after use
- Limit concurrent operations

🧪 Testing & Validation

Unit Tests (Conceptual)

```
php
// Test embedding generation
function testEmbedding() {
    $text = "Test text for embedding";
    $vector = embedText($text);

    assert(is_array($vector), "Embedding should return array");
    assert(count($vector) == 768, "Vector should be 768-dimensional");
    assert(is_float($vector[0]), "Vector elements should be floats");
}

// Test vector search
function testSearch() {
    $results = searchKnowledge("admission requirements");

    assert(is_array($results), "Results should be array");
    assert(count($results) <= 5, "Should return max 5 results");
```

```
    foreach ($results as $result) {
        assert(isset($result['content']), "Result should have content");
        assert(isset($result['score']), "Result should have score");
    }
}
```

Integration Tests

1. **Ollama Connection**: curl http://localhost:11434/api/tags
2. **Qdrant Health**: curl http://localhost:6333/collections
3. **PHP Endpoints**: curl -X POST http://localhost:8000/ask.php

Load Testing

- Simulate multiple concurrent users
- Measure response times
- Monitor memory usage

🔍 Troubleshooting Guide

Common Issues & Solutions

1. Ollama Not Responding

text

Error: "Ollama connection failed"

Solution:

1. Check if Ollama is running: `ollama list`

2. Start Ollama: `ollama serve`

3. Verify model is pulled: `ollama pull nomic-embed-text`

2. Qdrant Connection Issues

text

Error: "Qdrant connection failed"

Solution:

1. Check Docker: `docker ps` (should show qdrant)

2. Restart container: `docker restart qdrant`

3. Check port 6333: `curl http://localhost:6333`

3. Memory Exhaustion

text

Error: "Allowed memory size exhausted"

Solution:

1. Increase PHP memory: `ini_set('memory_limit', '512M')`

2. Process smaller chunks in ingest.php

3. Use `gc_collect_cycles()` in loops

4. Slow Response Times

text

Issue: Chat responses take >10 seconds

Solution:

1. Cache embeddings: Implement file caching

2. Reduce chunk size: Try 300 characters

3. Use lighter model: Switch to mistral:7b

5. Poor Search Results

text

Issue: Irrelevant answers

Solution:

1. Adjust similarity threshold: Try 0.6

2. Improve chunking: Use sentence boundaries

3. Add more context chunks: Increase from 5 to 7

Debug Mode

Enable in config.php:

php

```
define('DEBUG_MODE', true);

function debugLog($message) {
    if (DEBUG_MODE) {
```

```
        error_log("[MAKAI DEBUG] " . $message);
    }
}
```

📈 Scaling Considerations

Vertical Scaling

1. **Increase Ollama Workers:**

```
bash
OLLAMA_NUM_PARALLEL=4 ollama serve
```

2. **Qdrant Resource Allocation:**

```
bash
docker run -p 6333:6333 \
  --memory=4g \
  --cpus=2 \
  qdrant/qdrant
```

3. **PHP Optimization:**

```
ini
; php.ini settings
memory_limit = 1G
max_execution_time = 120
opcache.enable=1
```

Horizontal Scaling

1. **Multiple Ollama Instances:** Load balance embedding requests
2. **Qdrant Cluster:** Multi-node setup for high availability
3. **PHP Session Sharing:** Redis for distributed sessions

Database Scaling

- Implement sharding for large collections
- Use Qdrant's replication for redundancy
- Regular backup of vector data

🔒 Security Considerations

Local-Only Architecture Benefits

1. **Data Privacy:** All data stays on local machine
2. **No API Keys:** No external service dependencies
3. **Network Isolation:** No internet connection required

Additional Security Measures

```
php

// Input sanitization
function sanitizeInput($input) {
    $input = trim($input);
    $input = stripslashes($input);
    $input = htmlspecialchars($input, ENT_QUOTES, 'UTF-8');
    return $input;
}

// Session security
ini_set('session.cookie_httponly', 1);
ini_set('session.use_only_cookies', 1);
ini_set('session.cookie_secure', 1); // HTTPS only
```

File Upload Security (If adding upload feature)

- Validate file types
- Scan for malware
- Store outside web root
- Limit file size

📊 Monitoring & Logging

Logging System

```
php
```

```
class Logger {
    const LEVELS = ['DEBUG', 'INFO', 'WARN', 'ERROR'];

    public static function log($message, $level = 'INFO') {
        $timestamp = date('Y-m-d H:i:s');
        $logEntry = "[{$timestamp}] [{$level}] {$message}\n";

        // Write to file
        file_put_contents('logs/app.log', $logEntry, FILE_APPEND);

        // Console output in debug mode
        if (DEBUG_MODE && $level === 'DEBUG') {
            echo $logEntry;
        }
    }
}
```

Key Metrics to Monitor

1. **Response Time:** Average query latency
2. **Cache Hit Rate:** Embedding cache effectiveness
3. **Memory Usage:** PHP and Ollama memory consumption
4. **Error Rate:** Failed requests percentage
5. **User Engagement:** Questions per session

🚧 Known Limitations

Technical Limitations

1. **Model Context Window:** Limited to ~4000 tokens for most models
2. **Embedding Dimension:** Fixed at 768 (nomic-embed-text)
3. **Local Storage:** Vector DB limited by disk space
4. **Single User:** Designed for single local user

Functional Limitations

1. **No Real-time Updates:** Requires page refresh for new documents
2. **Basic Search:** No advanced filters or faceted search
3. **Limited File Formats:** Currently text-only, no PDF/Word
4. **No User Authentication:** Single anonymous session

🎯 Future Enhancements

Short-term (Next Version)

1. **PDF Support:** Add PDF parsing with pdf2text
2. **Web Crawler:** Ingest university website content
3. **Export Feature:** Export chat conversations
4. **Admin Dashboard:** Monitor system health

Medium-term

1. **Multi-language Support:** Add local language models
2. **Voice Interface:** Speech-to-text input
3. **Offline Mode:** Progressive web app capabilities
4. **Plugin System:** Extensible knowledge sources

Long-term

1. **Multi-user Support:** User accounts and permissions
2. **Advanced Analytics:** Question patterns and trends
3. **Mobile App:** iOS/Android applications
4. **API Service:** REST API for third-party integration

📚 Knowledge Base Best Practices

Document Preparation

1. **Clean Text:** Remove headers, footers, page numbers
2. **Consistent Formatting:** Use Markdown for structure
3. **Metadata:** Include document source and date
4. **Regular Updates:** Schedule monthly knowledge refreshes

Chunking Strategy

- **Academic Content:** Chunk by section/subsection
- **FAQ Documents:** One Q&A per chunk
- **Policy Documents:** Keep entire policies together
- **Mixed Content:** Hybrid approach based on content type

Quality Control

1. **Sample Queries:** Test with common questions
2. **Accuracy Check:** Manual review of AI responses
3. **Coverage Analysis:** Identify knowledge gaps
4. **Feedback Loop:** User feedback for improvement

🤝 Contributing Guidelines

Code Contributions

1. Fork the repository
2. Create feature branch: git checkout -b feature/amazing-feature
3. Commit changes: git commit -m 'Add amazing feature'
4. Push to branch: git push origin feature/amazing-feature
5. Open Pull Request

Documentation Contributions

- Update this README for changes
- Add comments to complex functions
- Create examples for new features
- Document edge cases

Testing Contributions

- Write unit tests for new features
- Test on different PHP versions
- Verify cross-browser compatibility
- Performance benchmarking

📄 License & Attribution

Open Source Components

- **Ollama:** MIT License
- **Qdrant:** Apache 2.0 License
- **Tailwind CSS:** MIT License
- **Markdown-it:** MIT License

Project License

This project is licensed under the MIT License - see the LICENSE file for details.

Attribution

Please credit:

- Ollama for local LLM inference
- Qdrant for vector search
- Makerere University for the use case

🙏 Acknowledgments

- **Makerere University** for inspiring the project
- **Ollama Team** for making local AI accessible
- **Qdrant Team** for the excellent vector database
- **Tailwind CSS** for the beautiful styling framework
- **PHP Community** for keeping the web simple

🆘 Getting Help

Support Channels

1. **GitHub Issues:** Bug reports and feature requests
2. **Documentation:** This README and code comments
3. **Community Forum:** (If established)
4. **Email Support:** (For enterprise deployments)

Quick Start Checklist

- Ollama running with models

- Qdrant Docker container active
- PHP server on port 8000
- Tailwind CSS built
- Knowledge documents added
- Ingestion script run

Common Commands Cheatsheet

bash

```
# Start all services
docker run -p 6333:6333 qdrant/qdrant &
ollama serve &
php -S localhost:8000 &

# Ingest documents
php ingest.php

# Monitor logs
tail -f logs/app.log

# Clear chat history
rm -rf sessions/*

# Rebuild CSS
npm run build
```

🏁 Conclusion

MAKAI demonstrates how to build a fully-functional, locally-running AI assistant using modern open-source tools. It solves the problem of providing instant, accurate information about Makerere University while maintaining complete data privacy and control.

The project successfully integrates:

- **Local AI inference** with Ollama
- **Efficient vector search** with Qdrant
- **Modern web interface** with Tailwind CSS
- **Scalable architecture** that can grow with needs

This implementation serves as both a practical tool for Makerere University and a reference architecture for building similar local-first AI applications in other domains.

Last Updated: $(date)
Version: 1.0.0
Maintainer: MAKAI Development Team