AI-Powered Teams Chatbot: Semantic Search with Source Citations

Table of Contents

1. Introduction
2. Project Objective
3. Technology Stack
4. System Architecture
5. AnythingLLM Integration
6. Citation Formatting Logic
7. Hosting and Deployment
8. Development Process
9. Microsoft Teams Integration
10. Known Issues and Fixes
11. Future Enhancements
12. Conclusion

Introduction

The AI Teams Chatbot is a Microsoft Teams-integrated assistant built using the Microsoft Bot Framework and AnythingLLM, an open-source large language model framework hosted on an AWS EC2 instance. It processes natural language queries, retrieves contextually relevant answers from preprocessed PDF documents, and displays responses with formatted source citations. The focus on PDFs aligns with their prevalence in enterprise settings (e.g., lecture notes, reports), and AnythingLLM's PDF collector extracts text and metadata for semantic search. This ensures transparency and trust by attributing answers to specific documents.

Project Objective

The chatbot was designed to:

- **Answer Queries Semantically**: Use AnythingLLM’s semantic document retrieval to provide accurate, meaning-based answers.
- **Display Clear Citations**: Present concise, deduplicated citations from PDF sources in a readable format.
- **Ensure Scalability**: Operate as a cloud-hosted solution with Cloudflare Tunnel for secure bot server exposure and AWS EC2 for AnythingLLM.

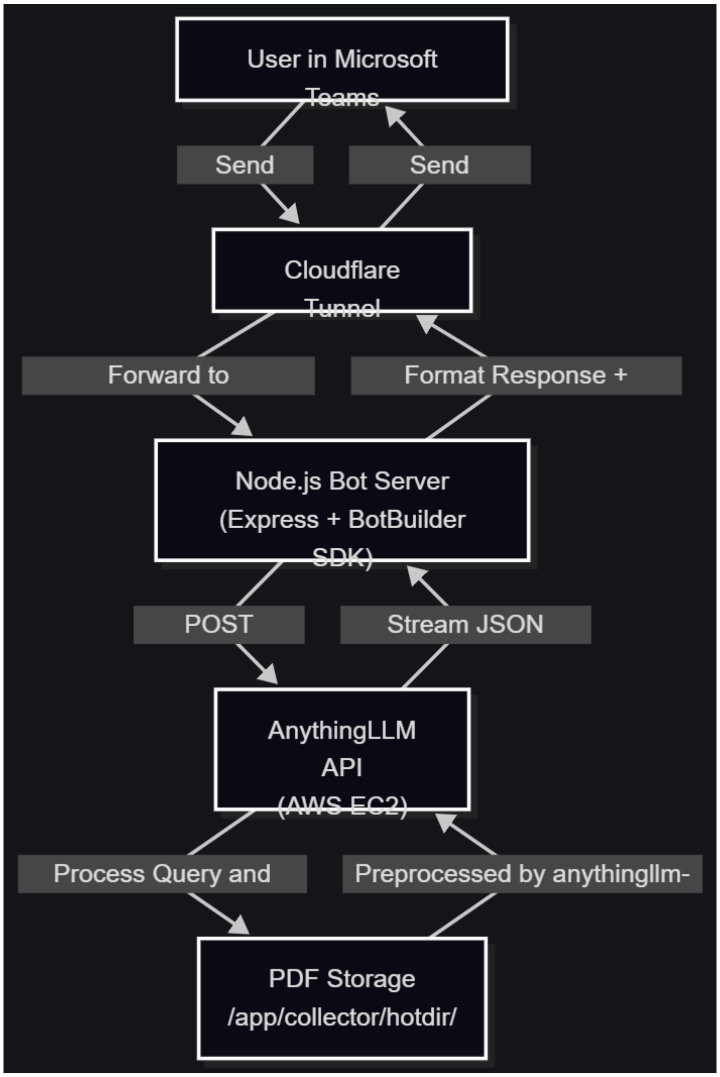
Technology Stack

Layer	Tools Used
Backend	Node.js (Express) – Scalable framework for bot logic and HTTP requests.
Bot Framework	Microsoft Bot Framework SDK – Enables Teams-compatible bot development.
Citation Engine	AnythingLLM (AWS EC2) – Handles semantic document retrieval and PDF processing.
Deployment	Cloudflare Tunnel (Bot) – Securely exposes local server; AWS EC2 (AnythingLLM).
Document Parsing	AnythingLLM PDF Collector – Extracts text/metadata and generates embeddings.

Why This Stack? Node.js ensures lightweight operations, the Bot Framework integrates seamlessly with Teams, AnythingLLM excels in enterprise-grade document retrieval, and Cloudflare Tunnel simplifies secure hosting. The PDF collector, part of AnythingLLM, uses anythingllm-cli to preprocess documents for semantic search.

System Architecture Diagram

The diagram below illustrates the data flow from a user query in Microsoft Teams to the final response with citations.


User in Microsoft Teams
Send
Send
Cloudflare Tunnel
Forward to
Format Response +
Node.js Bot Server (Express + BotBuilder SDK)
POST
Stream JSON
AnythingLLM API (AWS EC2)
Process Query and
Preprocessed by anythingllm-
PDF Storage /app/collector/hotdir/

A visual representation of the chatbot's architecture, showing the user in Microsoft Teams sending a query to a Cloudflare Tunnel, which forwards it to a Node.js Bot Server. The bot server sends a POST request to the AnythingLLM API on AWS EC2, which processes the query using preprocessed PDFs from storage. The API streams JSON chunks back to the bot server, which formats the response and sends it to Teams via the tunnel. A note indicates the Groq API key configuration in AnythingLLM's UI.

System Architecture

1. **User Query**: Sent via Microsoft Teams chat.
2. **Bot Endpoint**: Received at /api/messages (exposed via Cloudflare Tunnel).
3. **AnythingLLM Query**: Bot sends a POST request to AnythingLLM’s streaming API.
4. **Response Streaming**: API returns JSON chunks:
    - textResponseChunk: Incremental answer text.
    - finalizeResponseStream: Array of source citations.
5. **Response Processing**: Bot buffers text, deduplicates citations by title, and formats the response.
6. **User Response**: Final answer with citations is sent to Teams.

Streaming: Reduces latency by delivering partial answers in real-time.
Error Handling: Future versions should include fallback messages for API failures, with debugging via Azure and Cloudflare logs.

AnythingLLM Integration

Endpoint: POST http://<ec2-ip>:3001/api/v1/workspace/myworkspace/stream-chat
Headers:

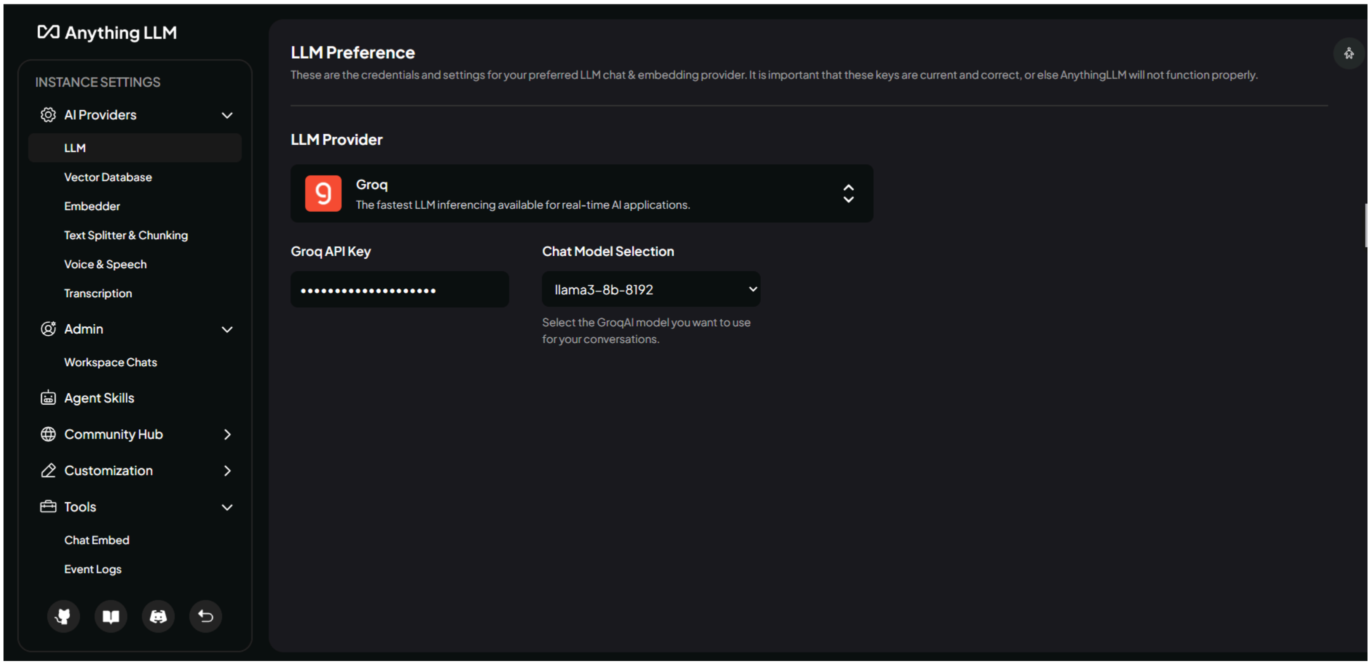
```
{

  "Content-Type": "application/json",

  "Authorization": "Bearer <API-KEY>"

}
```

Request Body:

```
{

  "message": "What is context switching?",

  "mode": "query",

  "userId": 1,

  "attachments": []
}
```

Response: Streamed JSON with textResponseChunk (answer text) and finalizeResponseStream (citations).
Configuration: A Groq API key, obtained from the Groq website (https://console.groq.com), was added to AnythingLLM's developer keys section via its UI to enable language model capabilities for query processing and embeddings.
Security: The AnythingLLM API key is stored in environment variables (e.g., .env file), and the Groq API key is managed securely within AnythingLLM's UI. Communication uses HTTPS via Cloudflare Tunnel.
Mode: query enables semantic retrieval, matching queries to document content by meaning.
Testing Note: The streaming response may close connections in Postman; use Node.js scripts or cURL for testing (see Known Issues and Fixes).

AnythingLLM UI Configuration



LLM Preference

These are the credentials and settings for your preferred LLM chat & embedding provider. It is important that these keys are current and correct, or else AnythingLLM will not function properly.

LLM Provider

Groq
The fastest LLM inferencing available for real-time AI applications.

Groq API Key
••••••••••••••••••••

Chat Model Selection
llama3-8b-8192
Select the GroqAI model you want to use for your conversations.

Citation Formatting Logic

Steps

1. Stream and decode JSON chunks.
2. Buffer response text until complete.

3. On finalizeResponseStream, extract unique sources by comparing title.
4. Format citations as:

- **<title>**
- Published: <date>
- > <snippet (max 200 chars)>
- Remove <document_metadata> tags (e.g., author info) using regex for clean snippets.
- Display as markdown in Teams for readability.
-

Details:

- **200-Character Limit**: Ensures concise citations in Teams' chat UI.
- **Metadata Removal**: Strips irrelevant tags for clarity.
- **Deduplication**: Prevents redundant citations from the same document.

Hosting and Deployment

Component	Hosting
Bot Server	Cloudflare Tunnel – Assigns secure public URL for local server.
AnythingLLM	AWS EC2 – Hosts Node.js, PDF Collector, and LLM embeddings.
Files	/app/collector/hotdir/ – Stores PDFs for preprocessing by anythingllm-cli.

Cloudflare Tunnel: Runs cloudflared daemon to expose the bot server securely without a public IP.
AnythingLLM Preprocessing: anythingllm-cli extracts text/metadata from PDFs and generates embeddings for semantic search.
Scalability: Development-ready; production requires EC2 scaling or Azure/ECS migration.

Development Process

1. Created Microsoft Azure account and registered a multi-tenant bot for cross-organization support.
2. Enabled Microsoft Teams channel in Azure.
3. Generated Microsoft App ID and Password.
4. Granted API permissions: Bot.Read, Bot.ReadWrite, Chat.ReadWrite.
5. Set messaging endpoint: https://<cloudflare-tunnel-url>/api/messages.
6. Configured AnythingLLM on AWS EC2, adding a Groq API key (obtained from https://console.groq.com) to the developer keys section via AnythingLLM's UI to enable language model functionality.
7. Built Node.js bot server with Express and BotBuilder SDK.
8. Integrated AnythingLLM's streaming API, adding logic for chunk decoding and citation formatting.
9. Tested in Azure's "Test in Web Chat" and Postman (noting streaming closes Postman connections, handled via Node.js scripts).
10. Exposed local server via Cloudflare Tunnel.
11. Connected bot to Teams and validated end-to-end flow.

Multi-Tenant: Supports multiple Teams tenants for scalability.
Postman Testing: Streaming responses close connections, expected behavior; validated with scripts.

Microsoft Teams Integration

Steps

1. **Azure Portal**:
   - Enable Teams channel and accept terms.
   - Set messaging endpoint: https://<cloudflare-tunnel-url>/api/messages.
2. **Manifest Creation**:
   - Create manifest.json using App Studio or manually with MicrosoftAppId, name, description, and messaging endpoint.
3. **Teams Deployment**:
   - Upload as a custom app in Teams: Apps > Manage your apps.
4. **Testing**:
   - Verify chat flow, answers, and citations in Teams.
5. **Debugging**:
   - Use Azure Logs for bot activity and Cloudflare Logs for tunnel issues.

Common Errors:

- **Invalid Endpoint**: Verify Cloudflare Tunnel URL.
- **Permission Issues**: Check API permissions in Azure.
- **Manifest Errors**: Validate JSON schema (e.g., correct MicrosoftAppId).

Known Issues and Fixes

Issue	Fix
No preview available	Extract preview from text field in sources.
Duplicate citations	Filter by title before rendering.
Token limit exceeded	Limit snippets to 200 characters.
Postman closes connection	Expected for streaming; test with Node.js scripts or cURL.

Additional Concerns:

- **Rate Limits**: Monitor EC2 for API throttling.
- **Invalid PDFs**: Validate files with anythingllm-cli to avoid preprocessing errors.
- **Teams Rendering**: Use basic markdown to avoid UI issues.

All issues are resolved; future versions could add cloud-hosted PDF links (e.g., AWS S3).

Future Enhancements

- **Collapsible Previews**: Implement Teams' adaptive cards with <details> or toggle actions (low complexity).
- **Card UI**: Display citations as expandable adaptive cards (moderate complexity).
- **Clickable Previews**: Host PDFs on AWS S3 or Azure Blob Storage for downloadable links (moderate complexity).

- **File Uploads**: Enable real-time uploads via Teams API and AnythingLLM’s attachments (high complexity).
- **Model Fallback**: Integrate OpenAI or Azure AI for reliability (high complexity).

These enhancements will improve user experience and system robustness.

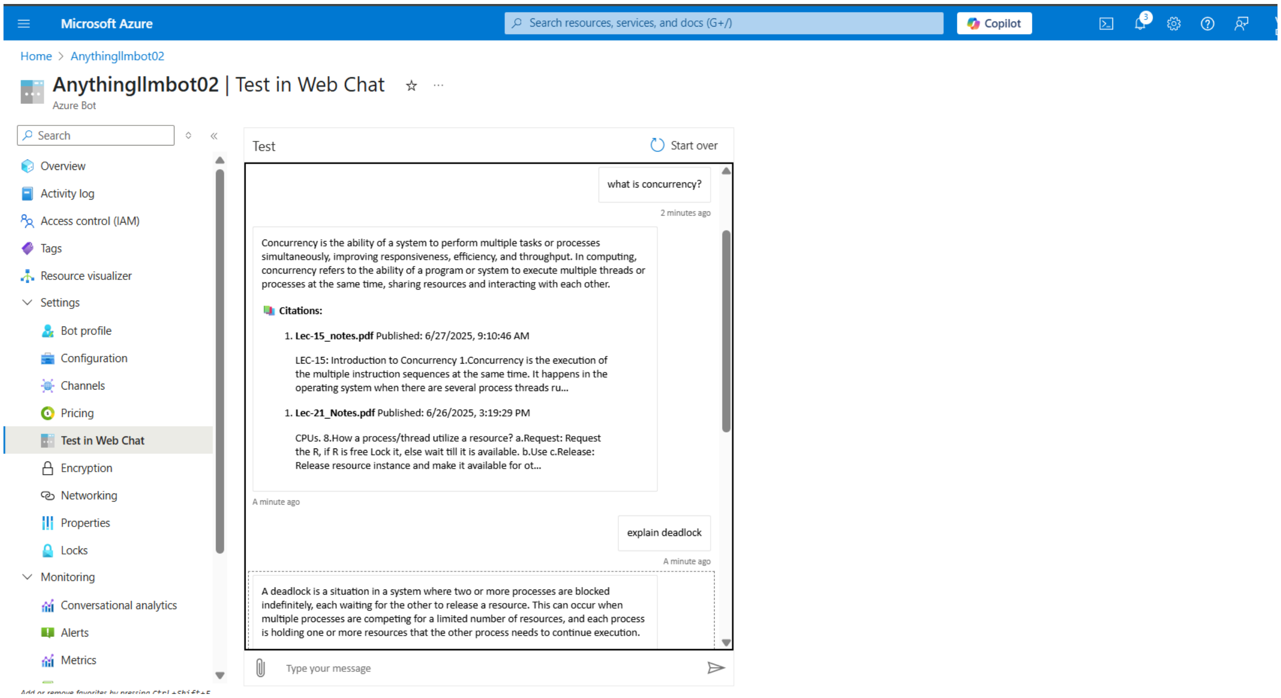
Conclusion

The AI Teams Chatbot combines semantic document retrieval with transparent source citations, making it a valuable tool for enterprise knowledge access. Its integration with Microsoft Teams, powered by AnythingLLM and hosted on AWS EC2, ensures scalability and reliability. While functional for development, production deployment requires scaling (e.g., EC2 Auto Scaling) and enhanced error handling.

Sample Response

Answer: Context switching is the process by which the operating system saves the current state of a thread or process and restores another.
Citations:

1. **Lec-15_notes.pdf**
   Published: 6/27/2025, 9:10:46 AM
   Doesn’t include switching of memory address space... Fast switching...
2. **Lec-21_Notes.pdf**
   Published: 6/26/2025, 3:19:29 PM
   Whenever a process requests a resource, it doesn’t hold any other...



what is concurrency?

2 minutes ago

Concurrency is the ability of a system to perform multiple tasks or processes simultaneously, improving responsiveness, efficiency, and throughput. In computing, concurrency refers to the ability of a program or system to execute multiple threads or processes at the same time, sharing resources and interacting with each other.

Citations:

1. **Lec-15_notes.pdf** Published: 6/27/2025, 9:10:46 AM

   LEC-15: Introduction to Concurrency 1.Concurrency is the execution of the multiple instruction sequences at the same time. It happens in the operating system when there are several process threads ru...

1. **Lec-21_Notes.pdf** Published: 6/26/2025, 3:19:29 PM

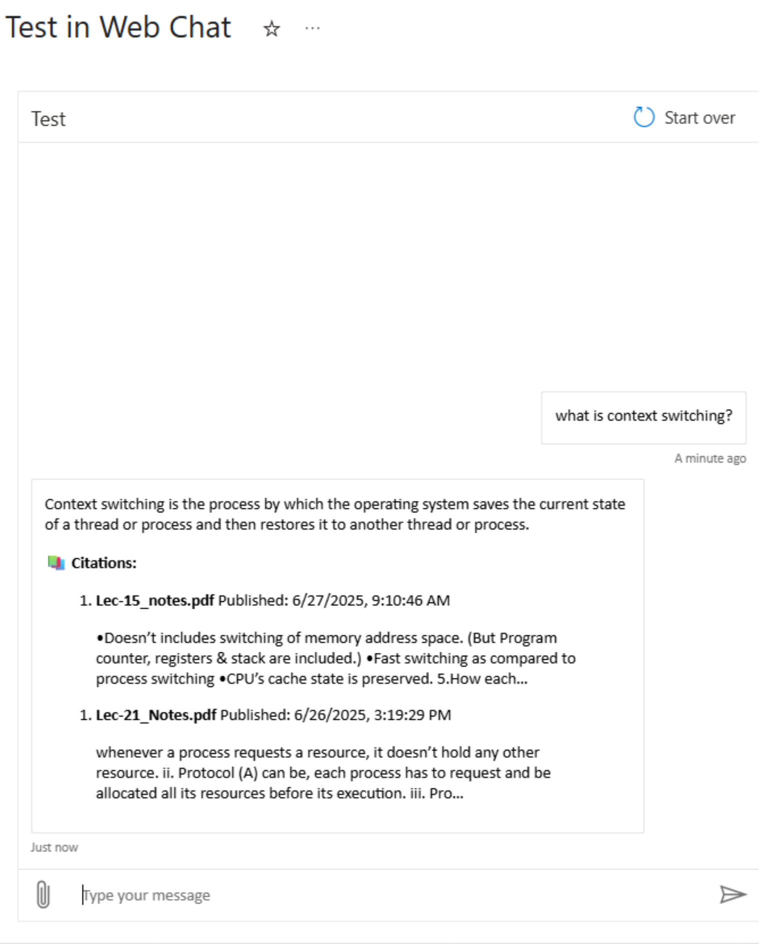
   CPUs. 8.How a process/thread utilize a resource? a.Request: Request the R, if R is free Lock it, else wait till it is available. b.Use c.Release: Release resource instance and make it available for ot...

A minute ago

explain deadlock

A minute ago

A deadlock is a situation in a system where two or more processes are blocked indefinitely, each waiting for the other to release a resource. This can occur when multiple processes are competing for a limited number of resources, and each process is holding one or more resources that the other process needs to continue execution.



what is context switching?

A minute ago

Context switching is the process by which the operating system saves the current state of a thread or process and then restores it to another thread or process.

Citations:

1. **Lec-15_notes.pdf** Published: 6/27/2025, 9:10:46 AM

   •Doesn't includes switching of memory address space. (But Program counter, registers & stack are included.) •Fast switching as compared to process switching •CPU's cache state is preserved. 5.How each...

1. **Lec-21_Notes.pdf** Published: 6/26/2025, 3:19:29 PM

   whenever a process requests a resource, it doesn't hold any other resource. ii. Protocol (A) can be, each process has to request and be allocated all its resources before its execution. iii. Pro...

Just now