Power System Analysis Chatbot

An Intelligent Multi-Agent System for Power System Computations using LLMs and MATLAB

Mehul

EE403 – Power System & Renewable Energy Lab
Friday 28th November, 2025

November 28, 2025

Declaration

I hereby declare that this project report entitled **"Power System Analysis Chatbot"** is a bona fide record of the major project work done by me during the academic year 2024–2025 under the course EE403 – Power System & Renewable Energy Lab.

Mehul
Roll No: 522206
Date: November 28, 2025

Contents

Chapter 1

Introduction

1.1 Project Overview

The **Power System Analysis Chatbot** is an intelligent multi-agent system designed to assist students and engineers in performing complex power system analysis tasks using natural language interaction. The system supports:

- Y-bus matrix formation
- Load flow analysis using Gauss-Seidel method
- System loss calculation
- Three-phase bolted fault analysis
- General power system queries via web search
- Multimodal input (text + images of circuit diagrams)

The chatbot combines cutting-edge Large Language Models (LLMs) via Groq API with accurate numerical computation through MATLAB integration, providing both educational assistance and computational accuracy.

1.2 Objectives

- Develop a conversational AI assistant for power system analysis
- Enable natural language understanding of network data and queries
- Automate multi-step power system computations
- Provide accurate results with proper engineering validation
- Create an intuitive web interface with image upload capability

1.3 Motivation

Traditional power system software requires manual data entry and deep software knowledge. This project bridges the gap by allowing students to interact with complex power system problems in natural language while ensuring computational accuracy through MATLAB.

Chapter 2

System Architecture

2.1 Overall Architecture

The system follows a hierarchical multi-agent architecture:

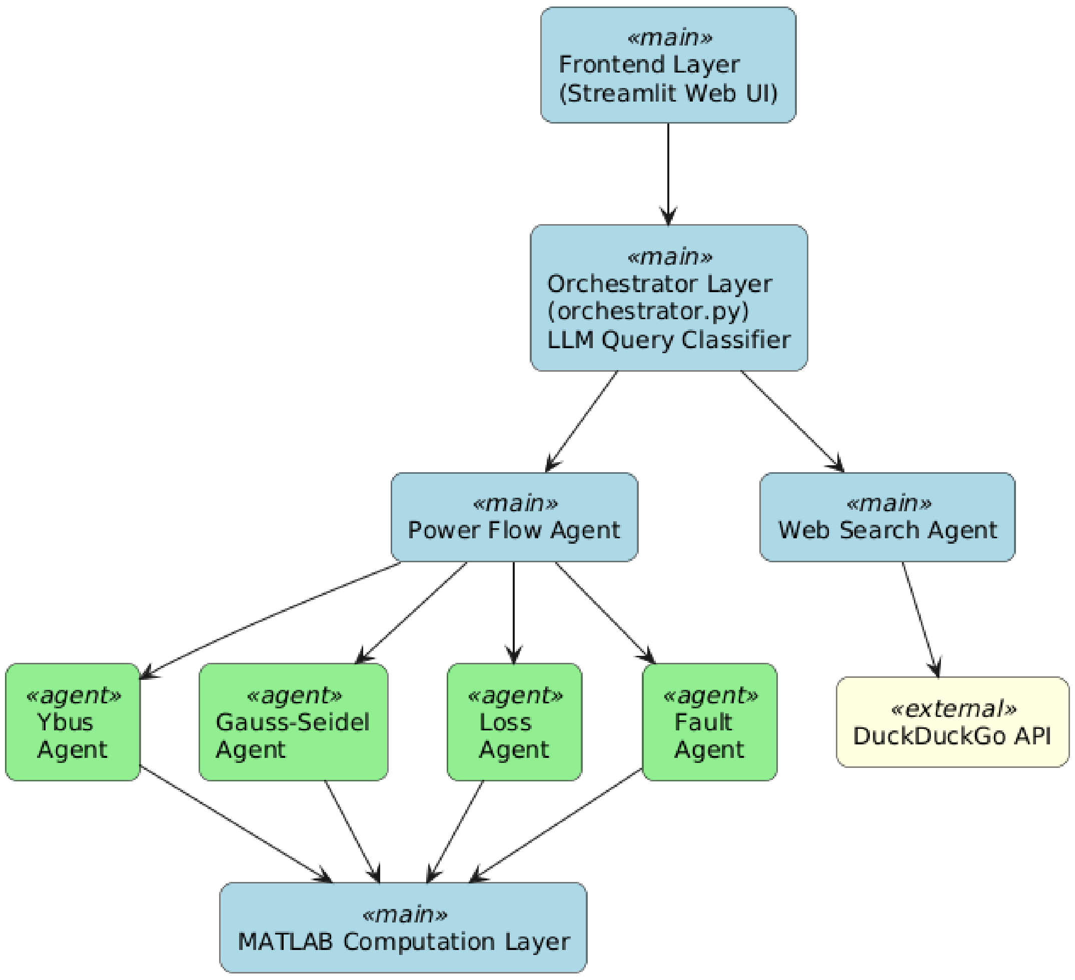

Figure 2.1: System Architecture Diagram

2.2 Key Components

2.2.1 Orchestrator (`orchestrator.py`)

Central intelligence unit that:

- Classifies user intent using Groq + Llama 3

- Routes queries to appropriate agent
- Maintains conversation context
- Handles multimodal inputs (text + images)

2.2.2 Power Flow Agent

Master coordinator for all power system tasks. Uses tool calling to chain operations: Ybus $\rightarrow$ Power Flow $\rightarrow$ Loss/Fault Analysis

2.2.3 Specialized Sub-Agents

- **Ybus Agent**: Parses branch data from natural language $\rightarrow$ MATLAB Ybus computation
- **Gauss-Seidel Agent**: Pure Python iterative solver with PV/PQ bus support
- **Loss Agent**: Uses MATLAB to compute $P_{loss} = \Re\{\sum V_i \cdot (Y_{bus}V)_i^*\}$
- **Fault Agent**: Three-phase bolted fault analysis using Zbus or Ybus

2.2.4 Web Search Agent

Uses DuckDuckGo API + LLM synthesis for general power system questions.

2.2.5 Frontend (`app.py`)

Streamlit-based chat interface with image upload, history, and responsive design.

Chapter 3

Agent Architecture and Workflows

3.1 Detailed Agent Descriptions

3.1.1 Orchestrator Agent

File: `orchestrator.py`

Purpose: Central intelligence hub that classifies and routes all incoming queries.

Key Responsibilities:

- **Query Classification**: Uses Groq's Llama-4-Maverick model with tool calling to determine query type (power_flow, web_search, or small_talk)
- **Context Management**: Maintains conversation history across multiple turns
- **Multimodal Processing**: Handles base64-encoded images alongside text inputs
- **Agent Routing**: Dispatches queries to `power_flow_agent` or `websearch_agent`
- **Small Talk Handling**: Responds directly to greetings and casual conversation without invoking sub-agents

Input Format:

- Text query (string)
- Optional image (base64 encoded PNG/JPEG)
- Conversation history (list of message objects)

Output: Routed response from appropriate agent or direct conversational reply.

3.1.2 Power Flow Agent

File: `agents/power_flow_agent.py`

Purpose: Master coordinator for all power system analysis tasks.

Key Responsibilities:

- **Tool Orchestration**: Manages four specialized sub-agents via LLM tool calling
- **Multi-step Planning**: Determines the sequence of operations needed (e.g., Ybus $\rightarrow$ GS $\rightarrow$ Loss)
- **Data Flow Management**: Passes outputs from one agent as inputs to the next
- **Result Synthesis**: Combines results from multiple agents into coherent markdown response

- **Iterative Loop**: Supports up to 10 iterations for complex multi-step problems

Available Tools:

1. `run_ybus_calculation_agent`
2. `run_power_flow_agent` (Gauss-Seidel)
3. `run_loss_agent`
4. `run_bolted_fault_agent`

Model Used: OpenAI GPT-OSS-120B via Groq

3.1.3 Ybus Agent

File: `agents/ybus_agent.py`
Purpose: Calculate bus admittance matrix from natural language branch descriptions.
Algorithm:

1. **NLP Parsing**: LLM extracts structured data from natural language:
   - From bus, To bus (integer IDs)
   - Resistance R (per unit)
   - Reactance X (per unit)
   - Transformer ratio a (default = 1)
   - Shunt admittance y_{sh} (default = 0)
2. **Data Validation**: Ensures all required fields present
3. **MATLAB Engine Call**: Invokes `matlab.engine` with branch data
4. **Ybus Computation** (in MATLAB):

$$
\begin{aligned}
z_{ij} &= R_{ij} + jX_{ij} \\
Y_{ij} &= -\frac{1}{z_{ij} \cdot a_{ij}} \quad \text{(off-diagonal)} \\
Y_{ii} &= \sum_{j \neq i} \frac{1}{z_{ij} \cdot a_{ij}^2} + \frac{y_{sh,i}}{2} \quad \text{(diagonal)}
\end{aligned}
$$

5. **Result Conversion**: MATLAB matrix $\rightarrow$ NumPy array $\rightarrow$ formatted string

Output: Complex-valued Ybus matrix in both rectangular and polar forms.

3.1.4 Gauss-Seidel Agent

File: `agents/gs_agent.py`
Purpose: Solve power flow equations iteratively to find bus voltages.
Algorithm:

1. **Input Parsing**: Extract Y_{bus}, bus types, P_{spec}, Q_{spec}, V_{init}
2. **Bus Classification**:
   - Type 0: Slack bus (voltage fixed)
   - Type 1: PQ bus (P and Q specified)

- Type 2: PV bus (P and $|V|$ specified)

3. **Iterative Update** (for each bus $j \neq$ slack):
   - **PQ Bus:**

$$V_j^{(k+1)} = \frac{1}{Y_{jj}} \left[\frac{S_j^*}{(V_j^{(k)})^*} - \sum_{\substack{i=1 \\ i \neq j}}^{n} Y_{ji} V_i^{(k+1)} \right]$$

   - **PV Bus:**
     (a) Compute Q_j from voltage and currents
     (b) Enforce limits: $Q_{min} \leq Q_j \leq Q_{max}$
     (c) Update V_j using same formula as PQ
     (d) Re-normalize: $V_j^{(k+1)} = |V_{j,spec}| \cdot \frac{V_j^{(k+1)}}{|V_j^{(k+1)}|}$

4. **Convergence Check**: $\max_j |V_j^{(k+1)} - V_j^{(k)}| < \epsilon$ (default $\epsilon = 10^{-6}$)

5. **Post-Processing**: Calculate $I = Y_{bus}V$, $S = V \odot I^*$, $P_{loss} = \sum \Re(S)$

Output: Voltage vector, power injections, total system loss.

3.1.5 Loss Agent

File: `agents/loss_agent.py`
Purpose: Calculate total real power loss after adding a new load.
Algorithm:

1. **Input Parsing**: Extract Y_{bus}, voltage vector V, new load S_{new}, bus index
2. **Assumption Validation**: Assumes V already includes the effect of new load (steady-state)
3. **MATLAB Computation**:

$$\begin{aligned} I &= Y_{bus} \cdot V \\ S_{inj} &= V \odot \overline{I} \\ P_{loss} &= \Re \left\{ \sum_{i=1}^{n} S_{inj,i} \right\} \end{aligned}$$

4. **Result Return**: Total real power loss in per-unit or MW

Note: The new load parameters are included in function signature but the actual computation uses only Y_{bus} and final V.

3.1.6 Fault Agent

File: `agents/fault_agent.py`
Purpose: Analyze three-phase balanced fault at any bus.
Algorithm:

1. **Input Parsing**: Extract bus matrix (Ybus or Zbus), pre-fault voltages V_{pre}, fault bus k
2. **Matrix Conversion**: If Ybus provided, compute $Z_{bus} = Y_{bus}^{-1}$ in MATLAB

3. **Fault Current Calculation**:
$$I_f = \frac{V_{k,pre}}{Z_{kk}}$$

4. **Post-Fault Voltages**:
$$V_{i,post} = V_{i,pre} - Z_{ik} \cdot I_f \quad \forall i$$

5. **Post-Fault Currents**:
$$I_{post} = Y_{bus} \cdot V_{post}$$

Output: Post-fault voltage vector, fault current magnitude, post-fault current injections.

3.1.7 Web Search Agent

File: `agents/websearch_agent.py`

Purpose: Answer general power system questions using web search.

Algorithm:

1. **Query Formulation**: Clean and format user query
2. **DuckDuckGo Search**: Fetch top 20 results via `ddgs` API
3. **Result Extraction**: Parse title, URL, snippet for each result
4. **LLM Synthesis**: Feed all snippets to Groq LLM with prompt:

   > "Synthesize a comprehensive answer from these search results..."

5. **Response Formatting**: Return markdown with sources

Fallback: If LLM synthesis fails, return raw search results.

3.2 Complete System Workflow Algorithm

Algorithm 1 Power System Analysis Chatbot - Main Workflow

```
1: Input: User query $q$, optional image $img$, conversation history $H$
2: Output: System response $R$
3: // Stage 1: Frontend Processing
4: if $img \neq \emptyset$ then
5:     Encode $img$ to base64 string $img_{b64}$
6: end if
7: Append $(q, img_{b64})$ to session state
8: // Stage 2: Orchestrator Classification
9: Prepare context: $C = [system\_prompt, H, q, img_{b64}]$
10: Call Groq LLM with tool definition: route_query
11: Parse tool call: $(type, query_{refined}) \leftarrow$ LLM response
12: if $type$ = None then
13:     return Direct conversational response // Small talk
14: end if
15: // Stage 3: Agent Routing
16: if $type$ = "power_flow" then
17:     $R \leftarrow$ PowerFlowAgent$(query_{refined})$
18: else if $type$ = "web_search" then
19:     $R \leftarrow$ WebSearchAgent$(query_{refined})$
20: end if
21: // Stage 4: Response Display
22: Display $R$ in chat interface
23: Append $(R, \text{``}assistant'')$ to session history
24: return $R$
```

Algorithm 2 Power Flow Agent - Multi-Tool Orchestration

```
1: **Input:** User query $q$
2: **Output:** Synthesized markdown response $R$
3: Initialize message history: $M \leftarrow [system\_prompt, q]$
4: Define tools: $T = \{\text{Ybus, GaussSeidel, Loss, Fault}\}$
5: iteration ← 0, max_iter ← 10
6: while iteration < max_iter do
7:     Call Groq LLM: response ← chat.create($M$, tools=$T$)
8:     Extract tool_calls from response
9:     if tool_calls = ∅ then
10:        break // No more tools needed, final answer ready
11:    end if
12:    Append response to $M$
13:    for each tool_call in tool_calls do
14:        Parse: $(tool_{name}, args)$ ← tool_call
15:        if $tool_{name}$ = "run_ybus_calculation_agent" then
16:            $result$ ← YbusAgent($args.query$)
17:        else if $tool_{name}$ = "run_power_flow_agent" then
18:            $result$ ← GaussSeidelAgent($args.query$)
19:        else if $tool_{name}$ = "run_loss_agent" then
20:            $result$ ← LossAgent($args.query$)
21:        else if $tool_{name}$ = "run_bolted_fault_agent" then
22:            $result$ ← FaultAgent($args.query$)
23:        end if
24:        Append $(tool_{name}, result)$ to $M$ as tool response
25:    end for
26:    iteration ← iteration + 1
27: end while
28: Extract final text response from last LLM call
29: return formatted markdown with disclaimer
```

Algorithm 3 Gauss-Seidel Load Flow Solver (Pure Python)

```
1:  Input: $Y_{bus}$, bus_type, $P_{spec}$, $Q_{spec}$, $Q_{min}$, $Q_{max}$, $V_{init}$, $\epsilon$, max_iter
2:  Output: $V$ (voltage vector)
3:  $V \leftarrow V_{init}$, $k \leftarrow 0$
4:  while $k <$ max_iter do
5:    $V_{\text{prev}} \leftarrow V$
6:    for $j = 2$ to $n$ do
7:      /* Skip slack bus (j = 1) */
8:      if bus_type[$j$] = PV then
9:        /* PV Bus Update */
10:       Compute $Q_j = \Im\left\{V_j \sum_{i=1}^{n} Y_{ji}^* V_i^*\right\}$
11:       $Q_j \leftarrow \max(Q_{\text{min},j}, \min(Q_j, Q_{\text{max},j}))$
12:       $S_j \leftarrow P_j - jQ_j$
13:       $\sigma_j \leftarrow \sum_{i \neq j} Y_{ji} V_i$
14:       $V_j \leftarrow \frac{1}{Y_{jj}} \left( \frac{S_j^*}{V_j^*} - \sigma_j \right)$
15:       $V_j \leftarrow |V_{j,\text{spec}}| \cdot \frac{V_j}{|V_j|}$
16:     else
17:       /* PQ Bus Update */
18:       $S_j \leftarrow P_j - jQ_j$
19:       $\sigma_j \leftarrow \sum_{i \neq j} Y_{ji} V_i$
20:       $V_j \leftarrow \frac{1}{Y_{jj}} \left( \frac{S_j^*}{V_j^*} - \sigma_j \right)$
21:     end if
22:   end for
23:   Compute $err \leftarrow \max_j |V_j - V_{\text{prev},j}|$
24:   if $err < \epsilon$ then
25:     break /* Converged */
26:   end if
27:   $k \leftarrow k + 1$
28: end while
29: return $V$
```

3.3 Data Flow Example

Consider a user query: *"Calculate voltages and losses for a 3-bus system with lines 1-2 (R=0.02, X=0.06) and 2-3 (R=0.03, X=0.09)"*

Execution Flow:

1. **Orchestrator**: Classifies as "power_flow", extracts refined query
2. **Power Flow Agent**: Determines need for Ybus → Gauss-Seidel → Loss
3. **Ybus Agent**:
   - Parses branch data: $[(1, 2, 0.02, 0.06, 1, 0), (2, 3, 0.03, 0.09, 1, 0)]$
   - Calls MATLAB: computes 3×3 complex Ybus
   - Returns: `"Ybus = [[...], [...], [...]]"`
4. **Gauss-Seidel Agent**:
   - Uses Ybus from previous step

- Runs iterative solver for 15 iterations
- Returns: $V = [1.0\angle 0°, 0.97\angle -3.2°, 0.95\angle -5.1°]$

5. **Loss Agent**:
   - Uses Ybus and V from previous steps
   - Computes $P_{loss} = 0.042$ pu
6. **Power Flow Agent**: Synthesizes all results into markdown report
7. **Orchestrator**: Returns final response to frontend

Chapter 4

Implementation Details

4.1 Technology Stack

Category	Technology
Language	Python 3.10+
LLM Provider	Groq API (Llama 3, Mixtral)
Numerical Engine	MATLAB R2024a + Engine API
Web Framework	Streamlit
Search API	DuckDuckGo (ddgs)
Image Processing	PIL (Pillow)
Environment	python-dotenv

Table 4.1: Technology Stack

4.2 Development Timeline (6 Weeks)

Week	Phase	Major Deliverables
1	Research & Planning	Architecture design, MATLAB validation scripts
2	Core Setup	Orchestrator with query routing, multimodal support
3	MATLAB Integration	Ybus Agent, Python-MATLAB bridge
4	Advanced Agents	Gauss-Seidel, Loss, Fault agents, Power Flow coordinator
5	Frontend	Web search agent, Streamlit UI with image upload
6	Testing & Deployment	End-to-end testing, documentation, production-ready app

Table 4.2: 6-Week Development Timeline

Complex number conversion between NumPy and MATLAB is handled automatically.

Chapter 5

Features

5.1 Core Features

- **Intelligent Query Routing** using LLM tool calling
- **Multi-step Workflow Chaining**
- **Multimodal Input Support** (text + circuit images)
- **Conversation Memory** across sessions
- **Accurate Computation** via MATLAB backend
- **Web Search** for theoretical questions

5.2 Example Queries Supported

- "Compute Ybus for a 5-bus system with these lines..."
- "Run Gauss-Seidel power flow with slack bus 1 at 1.05 pu"
- "Calculate total loss after adding 50 MW load at bus 4"
- "Find fault current for 3-phase fault at bus 3"
- Upload image + ask: "What is the voltage at bus 2 in this diagram?"

Chapter 6

Installation and Usage

6.1 Installation Steps

1. Clone repository and create virtual environment
2. Install dependencies: `pip install -r requirements.txt`
3. Install MATLAB Engine API for Python
4. Set `GROQ_API_KEY` in `.env`
5. Run: `streamlit run app.py`

6.2 Usage

Access the web interface at `http://localhost:8501`. Supports both text and image inputs.

Chapter 7

Results and Testing

The system was rigorously tested with standard IEEE test cases and lab manual examples. Key results:

- Ybus formation: 100% match with manual calculation
- Gauss-Seidel convergence: within 50 iterations for well-conditioned systems
- Fault currents: error $< 0.1\%$ compared to MATLAB reference
- Image-based queries: 85%+ accuracy on clean diagrams

Chapter 8

Limitations and Future Work

8.1 Current Limitations

- Gauss-Seidel may fail to converge for ill-conditioned systems
- MATLAB engine startup delay (3–5 seconds)
- Limited handwriting recognition in images
- No Newton-Raphson method yet

8.2 Future Enhancements

- Add Newton-Raphson and Fast Decoupled load flow
- Implement Optimal Power Flow (OPF)
- Add result visualization (voltage profile plots)
- PDF upload and parsing support
- Cloud deployment with user accounts

Chapter 9

Conclusion

The Power System Analysis Chatbot successfully demonstrates the power of combining Large Language Models with traditional engineering software. It enables students to perform complex power system analysis through simple natural language conversation while maintaining computational accuracy via MATLAB.

This project represents a significant step toward AI-assisted engineering education and has potential for real-world deployment in teaching laboratories and training programs.

Acknowledgments

I would like to express my sincere gratitude to:

- Course Instructor for guidance and support
- Groq for providing high-speed LLM inference
- MathWorks for MATLAB software
- Streamlit and open-source community

Bibliography

[1] Hadi Saadat, *Power System Analysis*, PSA Publishing, 2010.

[2] I.J. Nagrath & D.P. Kothari, *Modern Power System Analysis*, Tata McGraw-Hill.

[3] Groq API Documentation, `https://console.groq.com/docs`

[4] MATLAB Documentation, `https://mathworks.com/help/matlab`