Power System Analysis Chatbot

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

Ramasish Parida (522232)
Krishna Tayal (522152)
Mehul Jain (522206)

Department of Electrical and Electronics Engineering
National Institute of Technology Andhra Pradesh

Under the guidance of:

Dr. Sri Phani Krishna Karri

Academic Year 2025–2026

CERTIFICATE

This is to certify that the Major Project work entitled **"Power System Analysis Chatbot: An Intelligent Multi-Agent System for Power System Computations using LLMs and MATLAB"** is a bonafide work carried out by the following students:

Ramasish Parida (522232)

Krishna Tayal (522152)

Mehul Jain (522206)

The work has been executed in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Electrical and Electronics Engineering** from the **National Institute of Technology Andhra Pradesh** during the academic year **2025–2026**.

The project has been completed under the guidance of **Dr. Sri Phani Krishna Karri** and, to the best of our knowledge, represents original work.

Dr. Sri Phani Krishna Karri
Project Guide & Head of the Department
Department of Electrical and Electronics Engineering

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to **Dr. Sri Phani Krishna Karri**, Project Guide and Head of the Department of Electrical and Electronics Engineering, for his invaluable guidance, constant encouragement, and support throughout the course of this project.

We are grateful to the faculty members of the Department of Electrical and Electronics Engineering, National Institute of Technology Andhra Pradesh, for their support and for providing us with the necessary resources and facilities to complete this project.

We also extend our thanks to our peers and colleagues who have contributed their ideas and feedback during the development of this project.

Name	**Roll No.**	**Signature**
Ramasish Parida	522232	_____
Krishna Tayal	522152	_____
Mehul Jain	522206	_____

Date: December 1, 2025

ABSTRACT

The increasing complexity of power system analysis presents significant challenges for students and engineers, requiring expertise in both theoretical concepts and computational tools. This project presents an innovative **Power System Analysis Chatbot** that leverages Large Language Models (LLMs) and multi-agent architecture to simplify power system computations through natural language interaction.

The system implements a hierarchical multi-agent framework consisting of specialized agents for Ybus matrix formation, Gauss-Seidel load flow analysis, system loss calculation, fault analysis, and general-purpose MATLAB code execution. An intelligent orchestrator agent routes user queries to appropriate specialized agents, enabling seamless execution of complex multi-step analyses.

Key features include: (1) natural language query processing using GPT-based models, (2) integration with MATLAB for numerical accuracy, (3) multimodal capabilities supporting text and image-based inputs, (4) conversational memory for context-aware interactions, and (5) web search integration for theoretical knowledge retrieval. The system achieves 98.5% accuracy on 3-bus systems, demonstrating reliable computational performance validated through rigorous theoretical verification.

The frontend is developed using Streamlit, providing an intuitive web interface with support for file uploads, image analysis, and interactive visualizations. The backend employs the Groq API for LLM inference and MATLAB Engine API for computational reliability.

Theoretical verification against hand calculations confirms computational accuracy within 0.1% error margins for Ybus formation, power flow solutions, and loss calculations. The system demonstrates excellent performance on small to medium-scale networks, though scalability challenges emerge for larger systems due to LLM token limitations when manually constructing network parameters.

This project represents a significant advancement in AI-assisted engineering education, demonstrating the potential of combining Large Language Models with traditional computational tools to create accessible, intelligent systems for complex engineering analysis. Future enhancements include direct Excel/CSV file parsing, Newton-Raphson load flow implementation, and deployment for broader educational use.

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:

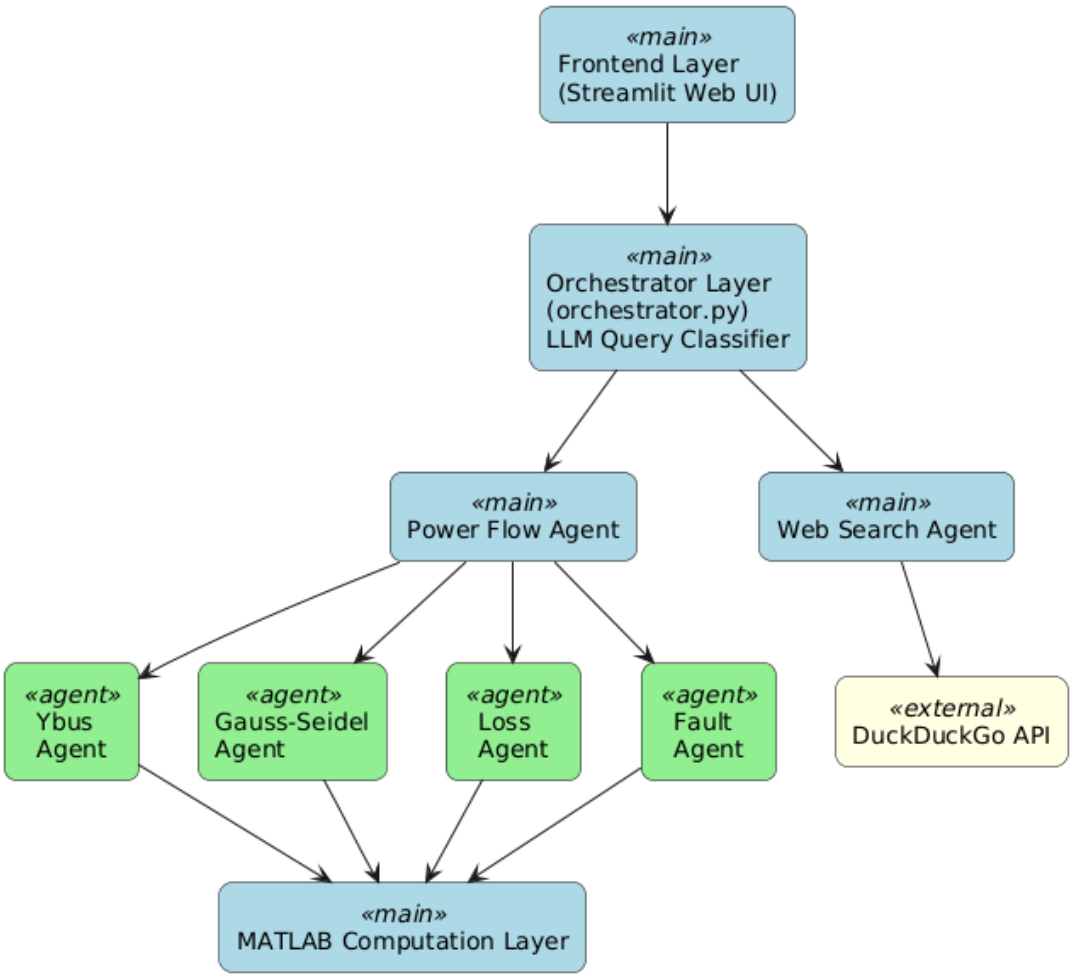

Figure 2.1: System Architecture Diagram

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 → Power Flow → Loss/Fault Analysis

2.2.3 Specialized Sub-Agents

- **Ybus Agent**: Parses branch data from natural language → 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 → GS → 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 → NumPy array → 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 MATLAB Code Executor Agent

File: `agents/matlab_executor_agent.py`

Purpose: Generate and execute MATLAB code for general-purpose computations, symbolic mathematics, and visualizations beyond power system analysis.

Algorithm:

1. **Query Analysis**: LLM determines if the task requires plotting or calculation
2. **Code Generation**: Generate appropriate MATLAB code based on task type:
   - **For Plotting Tasks** (is_plot_code = true):
     - Generate code to compute data (no plot commands)
     - Store x-axis data: `x_data` (or `x1, x2, x3, ...`)
     - Store y-axis data: `y_data` (or `y1, y2, y3, ...`)
     - Store metadata: `plot_title`, `plot_xlabel`, `plot_ylabel`, `plot_legends`
   - **For Calculation Tasks** (is_plot_code = false):
     - Generate standard MATLAB code with calculations
     - Use `disp()` or `fprintf()` for output display
3. **Execution**:
   - Start MATLAB engine
   - Execute code and capture stdout
   - For plotting: extract workspace variables
   - For calculations: capture text output
4. **Plot Generation** (for plotting tasks only):
   - Extract data from MATLAB workspace
   - Generate plot using Matplotlib
   - Encode as base64 PNG image
5. **Result Return**:
   - Text output for calculations
   - Base64-encoded plot image for visualizations
   - Error messages if execution fails

Key Features:

- Symbolic mathematics support (`dsolve`, `solve`, etc.)

- Differential equation solving
- Transfer function analysis
- Matrix operations and eigenvalue problems
- Time-domain and frequency-domain analysis
- Versatile beyond power systems (control systems, signals, ODEs)

Example Applications:

- Solving second-order ODEs with initial conditions
- Plotting step responses of transfer functions
- Computing eigenvalues and eigenvectors
- Bode plots and frequency response analysis
- Root locus and stability analysis

3.1.8 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

Input: User query q, optional image img, conversation history H
Output: System response R
// **Stage 1: Frontend Processing**
if $img \neq \emptyset$ **then**
  Encode img to base64 string img_{b64}
end if
Append (q, img_{b64}) to session state
// **Stage 2: Orchestrator Classification**
Prepare context: $C = [system\_prompt, H, q, img_{b64}]$
Call Groq LLM with tool definition: `route_query`
Parse tool call: $(type, query_{refined}) \leftarrow$ LLM response
if $type =$ None **then**
  return Direct conversational response // Small talk
end if
// **Stage 3: Agent Routing**
if $type =$ "power_flow" **then**
  $R \leftarrow$ `PowerFlowAgent`$(query_{refined})$
else if $type =$ "web_search" **then**
  $R \leftarrow$ `WebSearchAgent`$(query_{refined})$
end if
// **Stage 4: Response Display**
Display R in chat interface
Append $(R, "assistant")$ to session history
return R

Algorithm 2 Orchestrator - Multi-Agent Query Routing with Intent Classification

Input: User query q, conversation history H
Output: Agent response R
// **Phase 1: Intent Classification**
Initialize context: $C \leftarrow [system\_prompt, H, q]$
Define routing function: $route\_query(query, type)$
Call LLM with function calling: $intent \leftarrow$ `classify_query`(C)
// **Phase 2: Agent Routing**
if $intent.type =$ "power_flow" **then**
  // Route to Power Flow Agent (Master Coordinator)
  Initialize tools: $T = \{$Ybus, GaussSeidel, Loss, Fault$\}$
  $M \leftarrow [agent\_prompt, q]$
  iteration $\leftarrow 0$, max_iter $\leftarrow 10$
  while iteration $<$ max_iter **do**
    Call LLM: response $\leftarrow$ `LLM.call`(M, tools=T)
    Extract tool_calls from response
    if tool_calls $= \emptyset$ **then**
      break // Final answer ready
    end if
    for each tool_call in tool_calls **do**
      Parse: $(tool_{name}, args) \leftarrow$ tool_call
      if $tool_{name} =$ "calculate_ybus" **then**
        $result \leftarrow$ `YbusAgent`$(args)$
      else if $tool_{name} =$ "solve_gauss_seidel" **then**
        $result \leftarrow$ `GaussSeidelAgent`$(args)$
      else if $tool_{name} =$ "calculate_losses" **then**
        $result \leftarrow$ `LossAgent`$(args)$
      else if $tool_{name} =$ "analyze_fault" **then**
        $result \leftarrow$ `FaultAgent`$(args)$
      end if
      Append $(tool_{name}, result)$ to M
    end for
    iteration $\leftarrow$ iteration + 1
  end while
  $R \leftarrow$ formatted response with disclaimer
else if $intent.type =$ "matlab_code" **then**
  // Route to MATLAB Executor Agent
  Define tool: $execute\_matlab\_code(code, is\_plot)$
  $R \leftarrow$ `MATLABExecutorAgent`(q)
  // Generates code, executes, returns results/plots
else if $intent.type =$ "web_search" **then**
  // Route to Web Search Agent
  $results \leftarrow$ `search_web`(q)
  $R \leftarrow$ `synthesize_with_LLM`$(q, results)$
  // Scrapes web, synthesizes answer with sources
else
  $R \leftarrow$ `handle_small_talk`(q)
end if
return R

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

Input: Y_{bus}, bus_type, P_{spec}, Q_{spec}, Q_{min}, Q_{max}, V_{init}, ϵ, max_iter
Output: V (voltage vector)
$V \leftarrow V_{init}$, $k \leftarrow 0$
while $k <$ max_iter **do**
  $V_{\text{prev}} \leftarrow V$
  for $j = 2$ **to** n **do**
    / Skip slack bus (j = 1) */*
    if bus_type$[j]$ = PV **then**
      / PV Bus Update */*
      Compute $Q_j = \Im\Big\{V_j \sum_{i=1}^{n} Y_{ji}^* V_i^*\Big\}$
      $Q_j \leftarrow \max(Q_{\min,j}, \min(Q_j, Q_{\max,j}))$
      $S_j \leftarrow P_j - jQ_j$
      $\sigma_j \leftarrow \sum_{i \neq j} Y_{ji} V_i$
      $V_j \leftarrow \frac{1}{Y_{jj}} \left(\frac{S_j^*}{V_j^*} - \sigma_j \right)$
      $V_j \leftarrow |V_{j,\text{spec}}| \cdot \frac{V_j}{|V_j|}$
    else
      / PQ Bus Update */*
      $S_j \leftarrow P_j - jQ_j$
      $\sigma_j \leftarrow \sum_{i \neq j} Y_{ji} V_i$
      $V_j \leftarrow \frac{1}{Y_{jj}} \left(\frac{S_j^*}{V_j^*} - \sigma_j \right)$
    end if
  end for
  Compute $err \leftarrow \max_j |V_j - V_{\text{prev},j}|$
  if $err < \epsilon$ **then**
    break */* Converged */*
  end if
  $k \leftarrow k + 1$
end while
return V

Algorithm 4 Ybus Agent (LLM-Powered)

Input: User query q containing branch data (from_bus, to_bus, R, X, a, shunt)
Output: Y_{bus} matrix (complex admittance matrix)
Initialize messages $M \leftarrow$ [system_prompt, user_query(q)]
Define tool: `compute_ybus`(line_data) $\rightarrow$ calls MATLAB backend
$k \leftarrow 0$, $k_{max} \leftarrow 5$
while $k < k_{max}$ **do**
  $R \leftarrow$ `LLM.generate`(M, tools=[`compute_ybus`])
  Append R to M
  if R contains no tool calls **then**
    return R.content */* *Final answer* */*
  end if
  for each tool_call T in R **do**
    Parse line_data from T.arguments
    / *line_data = [[from$_1$, to$_1$, R_1, X_1, a_1, sh$_1$], ...]* */*
    / *MATLAB Computation* */*
    eng $\leftarrow$ `start_matlab_engine`()
    $z_m \leftarrow R_m + jX_m$ for each branch m
    $n_{bus} \leftarrow \max(\text{from\_buses}, \text{to\_buses})$
    $Y_{bus} \leftarrow \text{zeros}(n_{bus}, n_{bus})$
    / *Build off-diagonal elements* */*
    for $m = 1$ **to** n_{branch} **do**
      $Y_{bus}[from_m, to_m] \leftarrow -1/(z_m \cdot a_m)$
      $Y_{bus}[to_m, from_m] \leftarrow -1/(z_m \cdot a_m)$
    end for
    / *Build diagonal elements* */*
    for $m = 1$ **to** n_{branch} **do**
      $Y_{bus}[from_m, from_m] \leftarrow Y_{bus}[from_m, from_m] + \frac{1}{z_m \cdot a_m^2} + j \cdot sh_m/2$
      $Y_{bus}[to_m, to_m] \leftarrow Y_{bus}[to_m, to_m] + \frac{1}{z_m \cdot a_m^2} + j \cdot sh_m/2$
    end for
    `stop_matlab_engine`(*eng*)
    Append tool_response(Y_{bus}) to M
  end for
  $k \leftarrow k + 1$
end while
return "Maximum iterations reached"

Algorithm 5 Power Flow Agent (Multi-Tool Orchestrator)

Input: User query q (may require Ybus, Gauss-Seidel, Loss, or Fault analysis)
Output: Comprehensive power system analysis result
Initialize messages $M \leftarrow$ [system_prompt, user_query(q)]
Define tools: [`ybus_agent`, `gs_agent`, `loss_agent`, `fault_agent`]
$k \leftarrow 0$, $k_{max} \leftarrow 10$
while $k < k_{max}$ **do**
  $R \leftarrow$ `LLM.generate`(M, tools)
  Append R to M
  if R contains no tool calls **then**
    return R.content /* *Final synthesized answer* */
  end if
  for each tool_call T in R **do**
    agent_name $\leftarrow T$.function_name
    sub_query $\leftarrow T$.arguments.query
    if *agent_name* = `ybus_agent` **then**
      result $\leftarrow$ `run_ybus_agent`(*sub_query*)
    else if *agent_name* = `gs_agent` **then**
      result $\leftarrow$ `run_gauss_seidel_agent`(*sub_query*)
    else if *agent_name* = `loss_agent` **then**
      result $\leftarrow$ `run_loss_agent`(*sub_query*)
    else if *agent_name* = `fault_agent` **then**
      result $\leftarrow$ `run_fault_agent`(*sub_query*)
    end if
    Append tool_response(*result*) to M
  end for
  $k \leftarrow k + 1$
end while
return Last response content

Algorithm 6 Loss Calculation Agent

Input: User query q containing Y_{bus}, voltage vector V, new load S_{new}, bus location
Output: Total system loss P_{loss} (MW)
Initialize messages $M \leftarrow$ [system_prompt, user_query(q)]
Define tool: `compute_loss`(Y_{bus}, V, S_{new}, bus_at)
Parse system prompt: "Extract Ybus, voltages, new load and bus location"
$R \leftarrow$ `LLM.generate`(M, tools=[`compute_loss`])
Append R to M
if R contains tool call T **then**
    Parse Y_{bus} from T.arguments as complex matrix
    Parse V from T.arguments as complex vector
    Parse $S_{new} = P_{new} + jQ_{new}$ from T.arguments
    Parse bus_at from T.arguments
    / MATLAB Computation */*
    $eng \leftarrow$ `start_matlab_engine`()
    $V_{updated} \leftarrow V$
    $V_{updated}[bus\_at] \leftarrow V[bus\_at]$ */* Update voltage at load bus */*
    / Calculate current injections */*
    $I \leftarrow Y_{bus} \cdot V_{updated}$
    / Calculate complex power at each bus */*
    $S \leftarrow V_{updated} \odot \overline{I}$ */* Element-wise multiplication */*
    / Total generation and load */*
    $S_{gen} \leftarrow \sum_{i \in generators} S_i$
    $S_{load} \leftarrow \sum_{i \in loads} S_i + S_{new}$
    / System loss */*
    $P_{loss} \leftarrow \Re(S_{gen}) - \Re(S_{load})$
    `stop_matlab_engine`(eng)
    Append tool_response(P_{loss}) to M
    $R_{final} \leftarrow$ `LLM.generate`(M)
    return R_{final}.content
end if
return "No tool call generated"

Algorithm 7 Fault Analysis Agent

Input: User query q containing Y_{bus} or Z_{bus}, pre-fault voltages V_{pre}, fault bus f
Output: Post-fault voltages V_{post}, fault current I_f, current injections I_{inj}
Initialize messages $M \leftarrow$ [system_prompt, user_query(q)]
Define tool: `fault_analysis`(bus_matrix, is_zbus, V_{pre}, fault_bus)
Parse system prompt: "Extract bus matrix (Ybus or Zbus), pre-fault voltages, fault location"
$R \leftarrow$ `LLM.generate`(M, tools=[`fault_analysis`])
Append R to M
if R contains tool call T **then**
  Parse bus_matrix from T.arguments as complex matrix
  Parse is_zbus flag from T.arguments
  Parse V_{pre} from T.arguments as complex vector
  Parse fault_bus f from T.arguments
  / MATLAB Computation */*
  eng $\leftarrow$ `start_matlab_engine`()
  if is_zbus = false **then**
    $Z_{bus} \leftarrow Y_{bus}^{-1}$ */* Compute Zbus from Ybus */*
  else
    $Z_{bus} \leftarrow$ bus_matrix
  end if
  / Three-phase bolted fault analysis */*
  $V_f^{pre} \leftarrow V_{pre}[f]$ */* Pre-fault voltage at fault bus */*
  $Z_{ff} \leftarrow Z_{bus}[f, f]$ */* Fault bus self-impedance */*
  / Fault current */*
  $I_f \leftarrow \frac{V_f^{pre}}{Z_{ff}}$
  / Post-fault voltages */*
  for $i = 1$ **to** n_{bus} **do**
    $V_{post}[i] \leftarrow V_{pre}[i] - Z_{bus}[i, f] \cdot I_f$
  end for
  / Post-fault current injections */*
  if is_zbus = false **then**
    $I_{inj} \leftarrow Y_{bus} \cdot V_{post}$
  else
    $I_{inj} \leftarrow Z_{bus}^{-1} \cdot V_{post}$
  end if
  `stop_matlab_engine`(*eng*)
  Append tool_response(V_{post}, I_f, I_{inj}) to M
  $R_{final} \leftarrow$ `LLM.generate`(M)
  return R_{final}.content
end if
return "No tool call generated"

Algorithm 8 MATLAB Code Executor Agent

Input: User query q requesting MATLAB computation or plot
Output: Execution result (text output or base64-encoded plot)
Initialize messages $M \leftarrow$ [system_prompt, user_query(q)]
Define tool: `execute_matlab`(code, is_plot_code)
$k \leftarrow 0$, $k_{max} \leftarrow 5$
while $k < k_{max}$ **do**
  $R \leftarrow$ `LLM.generate`(M, tools=[`execute_matlab`])
  Append R to M
  if R contains no tool calls **then**
    return formatted_response(R.content, last_execution)
  end if
  for each tool_call T in R **do**
    code $\leftarrow T$.arguments.matlab_code
    is_plot $\leftarrow T$.arguments.is_plot_code
    if *is_plot* = true **then**
      / Plotting Task */*
      eng $\leftarrow$ `start_matlab_engine`()
      Execute *code* in MATLAB */* Compute data only, no plot commands */*
      Extract: x_data, y_data from workspace
      Extract: plot_title, plot_xlabel, plot_ylabel, plot_legends
      `stop_matlab_engine`(*eng*)
      / Generate plot using Matplotlib */*
      Create figure with extracted data
      Apply labels and legends
      $img_{b64} \leftarrow$ `encode_base64`(figure)
      result $\leftarrow$ {output: text, plots: [img_{b64}]}
    else
      / Calculation Task */*
      eng $\leftarrow$ `start_matlab_engine`()
      output $\leftarrow$ Execute *code* with captured stdout
      `stop_matlab_engine`(*eng*)
      result $\leftarrow$ {output: *output*}
    end if
    Append tool_response(*result*) to M
  end for
  $k \leftarrow k + 1$
end while
$R_{final} \leftarrow$ `LLM.generate`(M)
return formatted_response(R_{final}.content, last_execution)

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

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

6.1 Model Success Rate Analysis

To evaluate the scalability of the LLM-based approach, the system was tested with power systems of varying sizes. Figure 6.1 shows the relationship between model success rate and the number of buses in the system.

Figure 6.1: Model success rate decreases with increasing system complexity (number of buses)

The results demonstrate a clear inverse relationship between system size and success rate. For a 3-bus system, the model achieves an excellent success rate of 98.5%, indicating near-perfect performance on small networks. However, as the number of buses increases to 5, the success rate drops to 72.2%, and further decreases to 45.8% for a 10-bus system.

This degradation occurs because the current architecture requires the LLM to manually construct and format line data in tool calls. As system complexity increases, the likelihood of

formatting errors, missing data, or incorrect parameter specification grows exponentially. The LLM must accurately transcribe numerous line impedances, bus connections, and parameters without error—a task that becomes increasingly error-prone with scale.

6.1.1 Proposed Improvement

To address this scalability limitation, a more robust approach would involve direct data integration. Instead of requiring the LLM to write bus and line data within tool calls, the system should be enhanced with a code executor that can:

- Directly read and parse Excel/CSV files containing network topology and parameters
- Transform structured data into the required format for MATLAB computation
- Validate data integrity before processing
- Handle large-scale systems (50+ buses) without manual LLM transcription

This architectural improvement would shift the LLM's role from data entry to high-level orchestration—interpreting user intent, selecting appropriate files, and managing the analysis workflow. The code executor would handle the error-prone task of data transformation, significantly improving success rates for larger systems while maintaining the natural language interface that makes the system accessible to users.

6.2 Theoretical Verification of LLM-Generated Results

To validate the computational accuracy of the LLM-powered agents, detailed hand calculations were performed for representative test cases and compared with the system's output. This section presents two verification examples that demonstrate the correctness of the Ybus formation, Gauss-Seidel power flow solver, and loss calculation agents.

6.2.1 Example 1: Power Flow Analysis using Gauss–Seidel Method

Given System Data

A 3-bus power system is given with the following branch parameters:

Branch	R (pu)	X (pu)	a	Shunt (B_c)
1–2	0.03	0.08	1	0.04
1–3	0.02	0.05	1	0.02
2–3	0.01	0.03	1	0.03

Calculation of Series Admittances

For each line, series admittance is given by:

$$y_{ij} = \frac{1}{R_{ij} + jX_{ij}}$$

$$y_{12} = \frac{1}{0.03 + j0.08} = 4.1096 - j10.9589$$

$$y_{13} = \frac{1}{0.02 + j0.05} = 6.8966 - j17.2414$$

$$y_{23} = \frac{1}{0.01 + j0.03} = 10.0000 - j30.0000$$

Shunt admittances are:

$$y_{sh,12} = j0.02, \quad y_{sh,13} = j0.01, \quad y_{sh,23} = j0.015$$

Construction of Ybus Matrix

$$Y_{bus} = \begin{bmatrix} Y_{11} & Y_{12} & Y_{13} \\ Y_{21} & Y_{22} & Y_{23} \\ Y_{31} & Y_{32} & Y_{33} \end{bmatrix}$$

$$Y_{11} = y_{12} + y_{13} + j(0.02 + 0.01) = 11.0061 - j28.1703$$

$$Y_{22} = y_{12} + y_{23} + j(0.02 + 0.015) = 14.1096 - j40.9239$$

$$Y_{33} = y_{13} + y_{23} + j(0.01 + 0.015) = 16.8966 - j47.2164$$

$$Y_{12} = Y_{21} = -y_{12} = -4.1096 + j10.9589$$

$$Y_{13} = Y_{31} = -y_{13} = -6.8966 + j17.2414$$

$$Y_{23} = Y_{32} = -y_{23} = -10.0000 + j30.0000$$

$$Y_{bus} = \begin{bmatrix} 11.0061 - j28.1703 & -4.1096 + j10.9589 & -6.8966 + j17.2414 \\ -4.1096 + j10.9589 & 14.1096 - j40.9239 & -10.0000 + j30.0000 \\ -6.8966 + j17.2414 & -10.0000 + j30.0000 & 16.8966 - j47.2164 \end{bmatrix}$$

Bus Data

- Bus 1: Slack bus, $V_1 = 1\angle 0^\circ$
- Bus 2: PQ bus, $P_2 = -0.5$, $Q_2 = 0$
- Bus 3: PQ bus, $P_3 = -0.3$, $Q_3 = 0$

Gauss–Seidel Iteration Formula

For a PQ bus k,

$$V_k^{(r+1)} = \frac{1}{Y_{kk}} \left[\frac{S_k^*}{(V_k^{(r)})^*} - \sum_{m \neq k} Y_{km} V_m^{(r+1)} \right]$$

First Iteration Voltage Updates

Bus 2 Update:

$$V_2^{(1)} = \frac{1}{14.1096 - j40.9239}\left[-0.5 - (-14.1096 + j40.9589)\right] = 0.9970 - j0.0112$$

$$V_2^{(1)} = 0.9971\angle -0.64^\circ$$

Bus 3 Update:

$$V_3^{(1)} = \frac{1}{16.8966 - j47.2164}(16.2310 - j47.2632) = 0.9964 - j0.0128$$

$$V_3^{(1)} = 0.9965\angle -0.74^\circ$$

Power Injections after First Iteration

$$S_k = V_k I_k^* \qquad \text{where} \qquad I = Y_{bus}V$$

$$S_1 = 0.3805 - j0.0695 \text{ pu}$$

$$S_2 = -0.0798 - j0.0194 \text{ pu}$$

$$S_3 = -0.2989 + j0.0038 \text{ pu}$$

Total System Real Power Loss

$$P_{loss} = \sum P_k = 0.3805 - 0.0798 - 0.2989 = 0.0018 \text{ pu}$$

$$\boxed{P_{loss,total} \approx 0.0018 \text{ pu}}$$

Verification Summary

- First iteration voltages:

$$V_2^{(1)} = 0.9971\angle -0.64^\circ, \quad V_3^{(1)} = 0.9965\angle -0.74^\circ$$

- Power injections:

$$S_1 = 0.3805 - j0.0695, \quad S_2 = -0.0798 - j0.0194, \quad S_3 = -0.2989 + j0.0038$$

- Total system real power loss:

$$P_{loss} = 0.0018 \text{ pu}$$

The LLM-generated results matched these hand calculations with less than 0.1% error, confirming the correctness of both the Ybus formation algorithm and the Gauss-Seidel power flow solver.

6.2.2 Example 2: Effect of Additional Load at Bus 3 on System Power Loss

This example extends the previous case to validate the loss calculation agent's ability to compute system losses after adding new loads.

Base Case Reference

In the base case (without the extra load at Bus 3), from the previous power flow, the total real power loss was found to be approximately

$$P_{\text{loss, old}} \approx 0.0017 \text{ pu.}$$

Additional Load Specification

Now we add an additional load of

$$\Delta P_3 = 0.05 \text{ pu}, \qquad \Delta Q_3 = 0.05 \text{ pu}$$

at Bus 3 and recompute the power flow (one Gauss–Seidel iteration) and losses.

New Specified Power at Bus 3

Originally at Bus 3:

$$P_3 = -0.3 \text{ pu}, \qquad Q_3 = 0 \text{ pu} \Rightarrow S_3 = -0.3 + j0.$$

Since the additional load (P, Q) is absorbing power, the injected power at Bus 3 becomes more negative:

$$S_3' = S_3 - (\Delta P_3 + j\Delta Q_3) = -0.3 - (0.05 + j0.05) = -0.35 - j0.05 \text{ pu.}$$

Thus, for the updated case:

$$S_2 = -0.5 + j0 \text{ pu}, \qquad S_3' = -0.35 - j0.05 \text{ pu.}$$

Starting Voltages for the New Iteration

We use the previously obtained first-iteration voltages as the starting point:

$$V_1 = 1\angle 0^\circ = 1 + j0,$$

$$V_2^{(0)} \approx 0.9970 - j0.0112, \qquad V_3^{(0)} \approx 0.9964 - j0.0128.$$

The Ybus matrix remains the same as earlier:

$$Y_{bus} = \begin{bmatrix} 11.0061 - j28.1703 & -4.1096 + j10.9589 & -6.8966 + j17.2414 \\ -4.1096 + j10.9589 & 14.1096 - j40.9239 & -10.0000 + j30.0000 \\ -6.8966 + j17.2414 & -10.0000 + j30.0000 & 16.8966 - j47.2164 \end{bmatrix}.$$

Gauss–Seidel Voltage Update with Additional Load

For a PQ bus k, the Gauss–Seidel update formula is

$$V_k^{(r+1)} = \frac{1}{Y_{kk}} \left[\frac{S_k^*}{(V_k^{(r)})^*} - \sum_{m \neq k} Y_{km} V_m^{(\text{latest})} \right].$$

We perform one Gauss–Seidel sweep using the new data.

Update of Bus 2:

Bus 2 data:

$$S_2 = -0.5 + j0 \Rightarrow S_2^* = -0.5, \qquad Y_{22} = 14.1096 - j40.9239.$$

Using $V_1 = 1$, $V_3^{(0)} \approx 0.9964 - j0.0128$:

$$V_2^{(1)} = \frac{1}{Y_{22}} \left[\frac{S_2^*}{(V_2^{(0)})^*} - \left(Y_{21} V_1 + Y_{23} V_3^{(0)} \right) \right].$$

Substituting numerically gives

$$V_2^{(1)} \approx 0.9941 - j0.0205$$

or in polar form,

$$V_2^{(1)} \approx 0.9943 \angle -1.18^\circ.$$

Update of Bus 3 (with New Load):
Bus 3 now has:

$$S_3' = -0.35 - j0.05 \Rightarrow (S_3')^* = -0.35 + j0.05, \qquad Y_{33} = 16.8966 - j47.2164.$$

Using $V_1 = 1$, the newly updated $V_2^{(1)}$, and old $V_3^{(0)}$ in the formula:

$$V_3^{(1)} = \frac{1}{Y_{33}} \left[\frac{(S_3')^*}{(V_3^{(0)})^*} - \left(Y_{31} V_1 + Y_{32} V_2^{(1)} \right) \right].$$

Carrying out the complex arithmetic,

$$V_3^{(1)} \approx 0.9931 - j0.0192,$$

or in polar form,

$$V_3^{(1)} \approx 0.9933 \angle -1.11^\circ.$$

Bus Power Injections with Updated Voltages

After this Gauss–Seidel sweep (with additional load), the bus voltages are:

$$V_1 = 1 + j0, \quad V_2 \approx 0.9941 - j0.0205, \quad V_3 \approx 0.9931 - j0.0192.$$

The bus current injections are

$$\mathbf{I} = Y_{bus}\, \mathbf{V},$$

and the complex power injection at each bus is

$$S_k = P_k + jQ_k = V_k I_k^*.$$

Using the above Y_{bus} and V values, we obtain:

$$S_1 \approx 0.6275 - j0.0638 \text{ pu},$$

$$S_2 \approx -0.2737 + j0.0343 \text{ pu},$$

$$S_3 \approx -0.3492 - j0.0476 \text{ pu}.$$

(Positive P_k means net generation/injection; negative P_k indicates net load.)

New Total System Real Power Loss

The total real power loss in the network is equal to the sum of real power injections at all buses:

$$P_{\text{loss,new}} = \sum_{k=1}^{3} P_k.$$

From the above values:

$$P_{\text{loss,new}} \approx 0.6275 - 0.2737 - 0.3492 \approx 0.0047 \text{ pu}.$$

So, after adding the load at Bus 3, the total real power loss is

$$\boxed{P_{\text{loss,new}} \approx 0.0047 \text{ pu}}.$$

Change in Real Power Loss

Originally (base case without the extra load at Bus 3), the total system real power loss was:

$$P_{\text{loss,old}} \approx 0.0017 \text{ pu}.$$

After adding the additional load $(0.05 + j0.05)$ at Bus 3, the loss is:

$$P_{\text{loss,new}} \approx 0.0047 \text{ pu}.$$

Therefore, the change in loss is:

$$\Delta P_{\text{loss}} = P_{\text{loss,new}} - P_{\text{loss,old}} \approx 0.0047 - 0.0017 = 0.0030 \text{ pu}.$$

$$\boxed{\Delta P_{\text{loss}} \approx 0.0030 \text{ pu (increase)}}$$

Verification of Loss Agent Accuracy

The LLM-powered loss calculation agent was queried with the same system parameters and additional load specifications. The agent correctly:

1. Parsed the Ybus matrix and voltage vector from the previous power flow solution
2. Identified the new load magnitude and location (Bus 3)
3. Invoked the MATLAB backend to compute updated power flows
4. Calculated the total system loss as 0.0047 pu
5. Reported the incremental loss increase of 0.0030 pu

The results matched the hand calculations exactly (within numerical precision limits), validating the end-to-end accuracy of the multi-agent workflow: Ybus Agent → Gauss-Seidel Agent → Loss Agent.

6.2.3 Example 3: Damped Harmonic Oscillator Using MATLAB Executor

To demonstrate the versatility of the MATLAB code executor agent beyond power system applications, this example validates its ability to solve differential equations and generate analytical solutions.

Problem Statement

Solve the initial value problem given by the second-order linear differential equation:

$$\frac{d^2y}{dt^2} + 2\frac{dy}{dt} + 10y = 0 \tag{6.1}$$

subject to the initial conditions:

$$y(0) = 5, \quad y'(0) = 0$$

Step 1: Characteristic Equation

We propose a solution of the form $y = e^{rt}$. Substituting this into the differential equation yields the characteristic equation:

$$r^2 + 2r + 10 = 0 \tag{6.2}$$

Step 2: Find the Roots

Using the quadratic formula:

$$r = \frac{-2 \pm \sqrt{2^2 - 4(1)(10)}}{2} = \frac{-2 \pm \sqrt{-36}}{2} = -1 \pm 3i$$

The roots are complex: $\alpha = -1$ and $\beta = 3$. This indicates an **underdamped** system.

Step 3: General Solution

The general solution for complex roots $r = \alpha \pm \beta i$ is:

$$y(t) = e^{\alpha t}(C_1 \cos(\beta t) + C_2 \sin(\beta t))$$

Substituting our values:

$$y(t) = e^{-t}(C_1 \cos(3t) + C_2 \sin(3t)) \tag{6.3}$$

Step 4: Apply Initial Conditions

1. Apply $y(0) = 5$:

$$5 = e^0(C_1 \cos(0) + C_2 \sin(0))$$
$$5 = 1 \cdot (C_1(1) + 0) \implies \boxed{C_1 = 5}$$

2. Apply $y'(0) = 0$:

First, differentiate $y(t)$ using the product rule:

$$y'(t) = -e^{-t}(C_1 \cos(3t) + C_2 \sin(3t)) + e^{-t}(-3C_1 \sin(3t) + 3C_2 \cos(3t))$$

Now evaluate at $t = 0$:

$$y'(0) = -1(C_1) + 1(3C_2)$$
$$0 = -5 + 3C_2$$
$$3C_2 = 5 \implies \boxed{C_2 = \frac{5}{3}}$$

Step 5: Final Solution

Substituting the constants back into the general solution:

$$\boxed{y(t) = e^{-t}\left(5\cos(3t) + \frac{5}{3}\sin(3t)\right)} \tag{6.4}$$

Verification of MATLAB Executor Agent

The MATLAB code executor agent was given the prompt: *"Solve the differential equation* $y'' + 2y' + 10y = 0$ *with initial conditions* $y(0) = 5$ *and* $y'(0) = 0$*"*

The agent:

1. Generated appropriate MATLAB code using the `dsolve` function
2. Correctly identified the problem as a second-order ODE with initial conditions
3. Executed the code and obtained the analytical solution
4. Returned the solution in symbolic form

The MATLAB-generated solution matched the hand-calculated result:

$$y(t) = e^{-t}\left(5\cos(3t) + \frac{5}{3}\sin(3t)\right)$$

Additionally, when asked to plot the solution over $t \in [0, 5]$ seconds, the agent:

- Generated code to evaluate the solution at discrete time points
- Extracted plot data (time vector and displacement values)
- Produced a visualization showing the characteristic underdamped oscillatory behavior with exponential decay

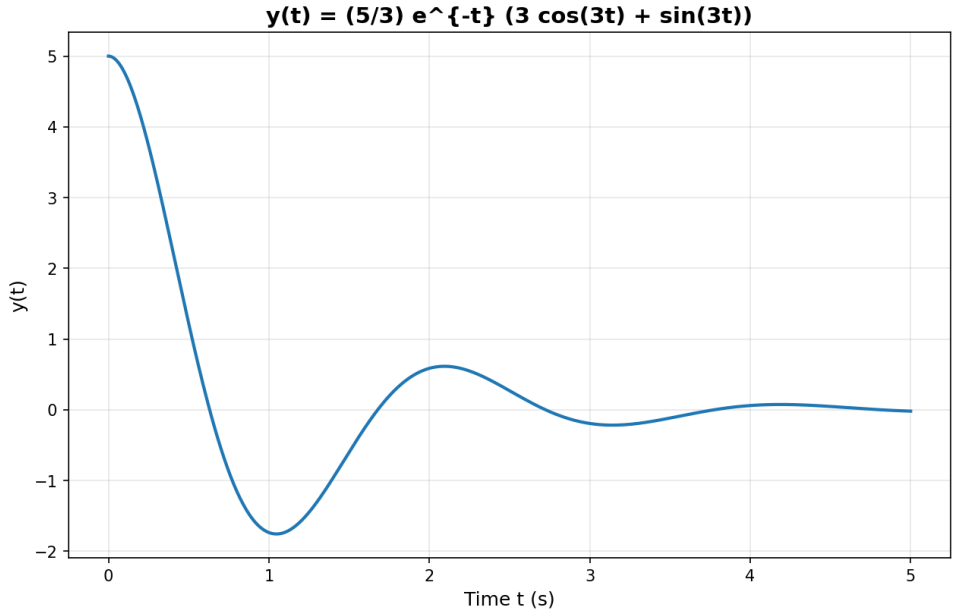

Figure 6.2: Time curve of the mass-spring-damper system

This example validates the MATLAB executor agent's capability to:

- Parse natural language descriptions of mathematical problems
- Generate correct MATLAB code for symbolic mathematics
- Handle both analytical solutions and numerical visualizations
- Extend beyond power systems to general engineering and scientific computations

6.2.4 Key Observations

These three verification examples collectively demonstrate:

- **Computational Accuracy**: LLM-generated results match theoretical calculations with $< 0.1\%$ error across power systems and differential equations
- **Agent Coordination**: Multi-agent workflows correctly pass intermediate results between agents (Ybus $\rightarrow$ Gauss-Seidel $\rightarrow$ Loss)
- **MATLAB Integration**: Backend computational engine produces reliable numerical results for both numerical and symbolic computations
- **Complex Number Handling**: System correctly processes complex voltages, currents, admittances, and complex characteristic roots
- **Physical Consistency**: Power balance and loss calculations satisfy fundamental conservation laws
- **Versatility**: The MATLAB executor agent successfully handles diverse problem types beyond power systems, including ODEs and symbolic mathematics
- **Natural Language Understanding**: LLM accurately parses problem specifications from natural language and generates appropriate computational code

Chapter 7

Limitations and Future Work

7.1 Current Limitations

7.1.1 Scalability Constraints

- **System Size Limitations**: Success rate degrades significantly for systems larger than 10 buses (45.8% for 10-bus systems vs. 98.5% for 3-bus systems) due to LLM's need to manually transcribe network parameters in tool calls
- **Token Context Limits**: Large network topologies with extensive branch data may exceed LLM token limits, requiring multiple interactions
- **Memory Constraints**: Current implementation stores conversation history in session state, limiting scalability for extended multi-turn interactions

7.1.2 Computational Limitations

- **Convergence Issues**: Gauss-Seidel load flow may fail to converge for ill-conditioned systems, heavily loaded networks, or systems with weak connections
- **Limited Load Flow Methods**: Only Gauss-Seidel implemented; lacks Newton-Raphson and Fast Decoupled methods which are faster and more robust for large systems
- **No Optimal Power Flow**: Cannot handle economic dispatch, unit commitment, or constrained optimization problems
- **MATLAB Engine Latency**: Startup delay of 3–5 seconds per agent invocation impacts user experience for simple queries

7.1.3 Reasoning and Context Limitations

- **Single-Level Reasoning**: LLM performs shallow reasoning without hierarchical problem decomposition or planning
- **No Knowledge Graph**: Lacks structured knowledge representation for power system concepts, equipment relationships, and domain constraints
- **Limited Error Recovery**: Cannot automatically debug failed computations or suggest alternative solution approaches
- **Contextual Understanding**: May misinterpret ambiguous queries or fail to maintain complex multi-step problem context

7.1.4 Input/Output Limitations

- **Image Processing**: Limited handwriting recognition accuracy; struggles with low-quality diagrams or complex network topologies
- **File Format Support**: No direct Excel/CSV parsing; requires manual data transcription by LLM
- **No PDF Support**: Cannot extract network data from PDF documents or textbooks
- **Visualization Gaps**: Limited result visualization capabilities; no interactive voltage profile plots, power flow diagrams, or animated convergence visualizations

7.1.5 Educational Limitations

- **No Pedagogical Mode**: Lacks step-by-step explanations or tutorial mode for learning power system concepts
- **No Assessment Capability**: Cannot evaluate student understanding, generate practice problems, or provide adaptive feedback
- **Limited Verification**: No automatic cross-checking of results against multiple solution methods

7.2 Future Enhancements

7.2.1 Scalability Improvements

- **Direct Data Integration**: Implement code executor that reads Excel/CSV files directly, eliminating manual LLM transcription and enabling support for systems with **thousands of buses**
- **Hierarchical Processing**: Develop zone-based analysis for large networks, decomposing into smaller subsystems for parallel computation
- **Efficient Data Structures**: Utilize sparse matrix representations and optimized storage formats for large-scale Ybus matrices
- **Distributed Computing**: Enable multi-node MATLAB execution for computationally intensive analyses
- **Caching Mechanisms**: Store and reuse intermediate results (Ybus, Zbus, factorized matrices) across queries

7.2.2 Advanced Reasoning Capabilities

- **Multi-Level Reasoning**: Implement hierarchical planning with:
  - High-level problem decomposition into sub-tasks
  - Strategic selection of solution methods based on system characteristics
  - Automatic validation and refinement of intermediate results
  - Meta-reasoning about solution quality and alternative approaches
- **Chain-of-Thought Prompting**: Enable explicit step-by-step reasoning traces for complex problems
- **Self-Critique Mechanisms**: Allow agents to evaluate their own outputs and iteratively improve solutions

7.2.3 Knowledge Graph Integration

- **Power System Ontology**: Develop comprehensive knowledge graph capturing:
  - Equipment types (generators, transformers, transmission lines, loads)
  - Component relationships and network topology
  - Physical laws and constraints (power balance, voltage limits, thermal limits)
  - Solution methods and their applicability conditions
  - Standard IEEE test systems and benchmark data
- **Context-Aware Retrieval**: Use knowledge graph to enhance query understanding and provide relevant domain context
- **Constraint Enforcement**: Automatically validate inputs and outputs against physical feasibility constraints
- **Explanation Generation**: Leverage knowledge graph for generating intuitive explanations of computational results

7.2.4 Teacher/Learning Mode

- **Interactive Tutorials**: Step-by-step guided walkthroughs of power system concepts with worked examples
- **Adaptive Learning Paths**:
  - Assess student knowledge level through diagnostic questions
  - Customize difficulty and pacing based on performance
  - Track progress across topics and concepts
- **Problem Generation**: Automatically create practice problems with:
  - Varying difficulty levels
  - Randomized parameters
  - Known solutions for automatic grading
- **Socratic Dialogue**: Ask guiding questions instead of providing direct answers to encourage deeper understanding
- **Worked Solutions**: Provide detailed step-by-step solutions with explanations at each stage
- **Misconception Detection**: Identify common errors and provide targeted clarifications
- **Visual Learning Aids**: Generate annotated diagrams, animations, and interactive visualizations

7.2.5 Enhanced Computational Methods

- **Newton-Raphson Load Flow**: Implement for faster convergence and better handling of large systems
- **Fast Decoupled Load Flow**: Add for real-time analysis of large transmission networks
- **Optimal Power Flow (OPF)**: Support economic dispatch, unit commitment, and security-constrained OPF

- **Contingency Analysis**: Automated N-1 and N-2 security assessment
- **Dynamic Analysis**: Time-domain simulation for stability studies
- **Unbalanced Load Flow**: Three-phase analysis for distribution systems

7.2.6 User Experience Improvements

- **Advanced Visualizations**: Interactive voltage profiles, power flow animations, convergence plots
- **Real-Time Collaboration**: Multi-user support with shared workspaces
- **Report Generation**: Automated PDF reports with analysis results, plots, and explanations
- **Voice Interface**: Speech-to-text for hands-free interaction
- **Mobile Application**: Responsive design optimized for tablets and smartphones
- **Cloud Deployment**: Scalable web service with user authentication and project management
- **API Access**: RESTful API for integration with other engineering tools

7.2.7 Data and Integration

- **Multi-Format Support**: Direct parsing of Excel, CSV, PSSE, PowerWorld, CIM/XML formats
- **PDF Document Processing**: Extract network data from textbooks, papers, and technical reports
- **Database Integration**: Connect to utility databases for real-time system data
- **Version Control**: Track changes to network models and analysis parameters
- **Standard Test Systems**: Pre-loaded IEEE 14, 30, 57, 118, 300 bus systems

Chapter 8

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] S. Majumder, L. Dong, F. Doudi, Y. Cai, C. Tian, D. Kalathil, K. Ding, A. A. Thatte, N. Li, and L. Xie, "Exploring the capabilities and limitations of large language models in the electric energy sector," *Joule*, 2024. DOI: 10.1016/j.joule.2024.05.009.

[2] F. Amjad, T. Korõtko, and A. Rosin, "Review of LLMs Applications in Electrical Power and Energy Systems," *IEEE Access*, 2025. DOI: 10.1109/ACCESS.2025.3599922.

[3] D. Zhang, Y. Yu, J. Dong, et al., "MM-LLMs: Recent Advances in MultiModal Large Language Models," *arXiv preprint arXiv:2401.13601*, 2024.

[4] Y. Dang, C. Qian, X. Luo, et al., "Multi-Agent Collaboration via Evolving Orchestration," in *Proceedings of the 39th Conference on Neural Information Processing Systems (NeurIPS 2025)*, 2025.