Software Requirement Specification (SRS)

AI-Powered Natural Language Linux Terminal Application

Submitted by:
Rishabh Kumar Jain(2301175), Rahul Yadav(2301168), Prakshay Saini(2301149)
Computer Science and Engineering
Indian Institute of Information Technology, Guwahati

Academic Year: 2025–2026

Contents

1. Introduction

1.1 Purpose of this Document

This Software Requirement Specification (SRS) document provides a detailed description of the requirements for the AI-Powered Natural Language Linux Terminal Application. The system enables users to interact with a Linux environment using natural language instead of traditional command-line syntax. The application opens as a terminal-style interface where users can type commands conversationally. This document serves as a reference for developers, testers, and evaluators.

1.2 Intended Audience

This document is intended for software developers, testers, academic evaluators, project supervisors, and future maintainers who require a complete understanding of the system's functionality and design.

1.3 Scope of the System

The system is a standalone terminal-style application developed in Python. When launched, it displays a terminal-like interface where users can directly begin typing instructions in natural language. A Large Language Model (LLM) processes the input, converts it into Linux shell commands, and sends them to a safety module for validation. Safe commands are then executed, and results are displayed within the same interface. The project demonstrates the integration of Artificial Intelligence with system-level command execution in a safe and user-friendly way.

1.4 Definitions, Acronyms, and Abbreviations

- **LLM** – Large Language Model
- **CLI** – Command Line Interface
- **GUI** – Graphical User Interface
- **API** – Application Programming Interface
- **NLP** – Natural Language Processing

1.5 Document Overview

This document describes the system architecture, features, operating environment, functional requirements, non-functional requirements, constraints, performance criteria, and future enhancements.

2. Overall Description

2.1 Product Perspective

The AI-Powered Terminal is a standalone desktop application that behaves like a terminal emulator but is enhanced with Artificial Intelligence. Unlike traditional terminals that require users to know specific commands, this system allows interaction using natural language. Internally, the application translates these instructions into executable Linux commands and runs them securely.

2.2 Product Functions

- Accept natural language instructions through a terminal-style interface.
- Convert natural language input into valid Linux shell commands using an LLM.
- Validate commands using a safety filtering module before execution.
- Execute validated commands in the Linux shell using controlled subprocess execution.
- Display command output and errors within the application window.
- Maintain session history including user inputs, generated commands, and execution outputs.

2.3 User Classes and Characteristics

- **Beginner Users**: Individuals unfamiliar with Linux commands who benefit from plain language interaction.
- **Intermediate Users**: Users who know basic commands but prefer faster task execution.
- **Advanced Users**: Experienced users who use the AI assistance for automation and efficiency.

2.4 Operating Environment

- Linux-based operating system (Ubuntu recommended)
- Python 3.x runtime environment
- Internet connection for LLM API communication
- Standard desktop or laptop hardware

2.5 Design and Implementation Constraints

- The system depends on the availability of the external LLM API.
- The application must prevent execution of harmful system-level commands.
- Execution must remain within normal user permissions.
- The interface must resemble a traditional terminal for user familiarity.

3. System Features

3.1 Terminal-Style User Interface

The application launches with a graphical window designed to resemble a Linux terminal. Users can type commands directly into the interface, and outputs appear in real time within the same window.

3.2 Natural Language Command Input

Users provide instructions in everyday English rather than standard Linux syntax.

3.3 LLM-Based Command Generation

The system sends user instructions to an AI model that converts them into shell commands accurately and contextually.

3.4 Command Safety Filter

Before execution, generated commands are analyzed for dangerous operations. Commands involving system deletion, shutdown, or unauthorized access are blocked.

3.5 Command Execution Engine

Safe commands are executed using Python's subprocess module in a controlled environment.

3.6 Output Rendering

Command outputs and errors are displayed clearly inside the terminal-style interface.

3.7 Session History

All commands and outputs are stored temporarily during the session for reference.

4. Functional Requirements

4.1 Input Processing

- The system shall accept natural language input through a terminal-style GUI.
- The system shall allow continuous command interaction.

4.2 AI Processing

- The system shall send user input to an LLM API.
- The system shall extract executable Linux commands from the AI response.

4.3 Safety Module

- The system shall validate commands using predefined safety rules.
- The system shall block execution of unsafe commands.

4.4 Execution Module

- The system shall execute validated commands in the system shell.
- The system shall capture standard output and errors.

4.5 Logging

- The system shall maintain session logs of interactions.

4.6 Command Recommendation

- The system shall recommend relevant commands following the origianl intention. for example - if the user ran ”Make folder named xyz” the system should recommend ”Open folder xyz”.

5. Performance Requirements

- AI command generation should complete within few seconds.
- The application should remain responsive during long sessions.
- System resource usage should remain moderate.

6. Security Requirements

- The system shall block destructive commands.
- API keys must be stored securely using environment variables.
- Command execution shall be restricted to user-level privileges.

7. Non-Functional Requirements

7.1 Usability

The interface must resemble a standard terminal while remaining beginner-friendly.

7.2 Reliability

The system should handle API or network failures gracefully.

7.3 Maintainability

The code should be modular and well-structured.

7.4 Portability

The application should function across major Linux distributions.

8. Future Enhancements

- Offline AI model integration
- Multi-command context handling
- Customizable themes for terminal interface
- Voice Command Feature can be integrated

9. Conclusion

The AI-Powered Natural Language Linux Terminal Application combines Artificial Intelligence with a terminal-style graphical interface to simplify Linux interaction. The system improves usability while maintaining security and performance, demonstrating an innovative application of AI in operating system interaction.