PROJECT AND TEAM INFORMATION

Project Title

(Try to choose a catchy title. Max 20 words).

NeuroShell: Blending Unix Power with Machine Learning Intelligence

Student / Team Information

Team Name:	*Kernel Mind*
Team member 1 (Team Lead) *(First name, Last Name, University Roll Number, student ID, email, picture)*	 *Anshika Saklani-2318420* *Student ID : 23012076* *Email :* *anshikasaklani894@gmail.com*
Team member 2 *(First name, Last Name, University Roll Number, student ID, email, picture)*	 *Rakhi Bisht-2319380* *Student Id : 23012177* *Email :* *rakhibisht24122004@gmail.com*

Team member 3 *(First name, Last Name, University Roll Number, student ID, email, picture)*	 *Akriti Rawat – 2318263* *Student Id: 230221541* *Email :* *akritirawat12345@gmail.com*
Team member 4 *(First name, Last Name, University Roll Number, student ID, email, picture)*	 *Ayush Chand – 2318582* *Student Id: 230112536* *Email :* *ayushchand862@gmail.com*

PROPOSAL DESCRIPTION (10 pts)

Motivation (1 pt)

(Describe the problem you want to solve and why it is important. Max 300 words).

Working in a Unix-like shell is powerful but not always beginner-friendly. Even experienced developers often face issues such as mistyped commands (gti instead of git), forgetting complex flags, or repeatedly typing long commands without assistance. These small inefficiencies add up to wasted time, reduced productivity, and frustration.
*Traditional shells like **bash** or **zsh** are excellent at command execution but lack intelligent guidance. While some shells offer simple autocompletion, they do not adapt to user habits, contexts, or past behavior.*

*Our motivation is to **bridge this gap by creating a smarter, self-learning shell** that not only runs commands but also acts as a supportive assistant. By combining **low-level OS functionality (C, POSIX)** with **machine learning models (Python)**, we aim to build a system that:*

- ***Reduces errors** by suggesting corrections for mistyped commands.*
- ***Saves time** by predicting the next likely command based on history and context.*
- ***Improves usability** by recommending flags and templates tailored to the user's workflow.*
- ***Empowers learning** by explaining why a suggestion was made*

State of the Art / Current solution (1 pt)

(Describe how the problem is solved today (if it is). Max 200 words).

*Currently, developers use shells like **bash**, **zsh**, and **fish**. While these shells are powerful for command execution and scripting, their assistance features remain limited. For example, zsh supports autocompletion, and fish provides syntax highlighting and history-based suggestions. However, these are mostly **rule-driven** and not context-aware.*

Remembering complex flags or repetitive commands often requires checking documentation or saving aliases, which adds manual overhead.

*Overall, today's solutions lack **self-learning, predictive, and explainable intelligence**. They do not adapt dynamically to user patterns (e.g., frequent Git usage in a repo), nor do they provide rationales for their suggestions. This leaves a gap for a shell that can combine **low-level OS functionality** with **AI-driven guidance** in an integrated, adaptive, and user-friendly way*

Project Goals and Milestones (2 pts)

(Describe the project general goals. Include initial milestones as well any other milestones. Max 300 words).

Our project aims to:

1. ***Develop a Unix-like shell core** in C (POSIX) with command parsing, job control, and built-in commands.*
2. ***Integrate a machine learning–based suggestion engine** in Python for typo correction, next-command prediction, and flag/template recommendations.*
3. ***Enable context-aware assistance**, where suggestions adapt to working directory, Git repository, and recent command history.*
4. ***Ensure self-learning and offline operation**, improving continuously from local history without cloud dependency.*
5. ***Provide explainable suggestions**, so users understand why a recommendation was made, increasing trust and usability.*

Project Approach (3 pts)

(Describe how you plan to articulate and design a solution. Including platforms and technologies that you will use. Max 300 words).

*Our solution is designed as a **two-component system** with a clear separation of concerns:*

1. ***Shell Core (Systems Component – C, POSIX)***
    - *We will implement the shell core in **C** using POSIX standards to ensure portability across Linux systems.*
    - *The shell will include:*
        - *A **REPL loop** for input/output handling.*
        - *A **parser** supporting commands, arguments, pipes (|), redirections (<, >, >>), and sequencing (;).*
        - ***Process management** using fork() and execvp().*
        - ***Job control** with foreground/background execution, signal handling (SIGINT, SIGTSTP, SIGCHLD), and built-ins (cd, exit, export, alias, history, etc.).*
        - *A **SQLite-based history store**, capturing timestamps, working directory, exit status, and command tags for later analysis.*
    - *Communication with the suggestion engine will occur via **Unix domain sockets**, exchanging newline-delimited JSON messages.*
2. ***Suggestion Engine (ML Component – Python)***

- *Built in **Python 3.10+**, this component will process live keystrokes and command context received from the shell.*
- *It will combine three suggestors:*
    - ***Typo Fixer** using Damerau-Levenshtein distance (rapidfuzz or custom implementation).*
    - ***Next Command Predictor** via N-gram/Markov models trained on SQLite history.*
    - ***Flag/Template Recommender** using TF-IDF and cosine similarity over past commands.*
- *A **lightweight ranker** will merge outputs and return top suggestions with confidence scores and rationales.*
- *The engine will operate entirely offline, ensuring privacy and reliability.*

***Platforms and Technologies**:*

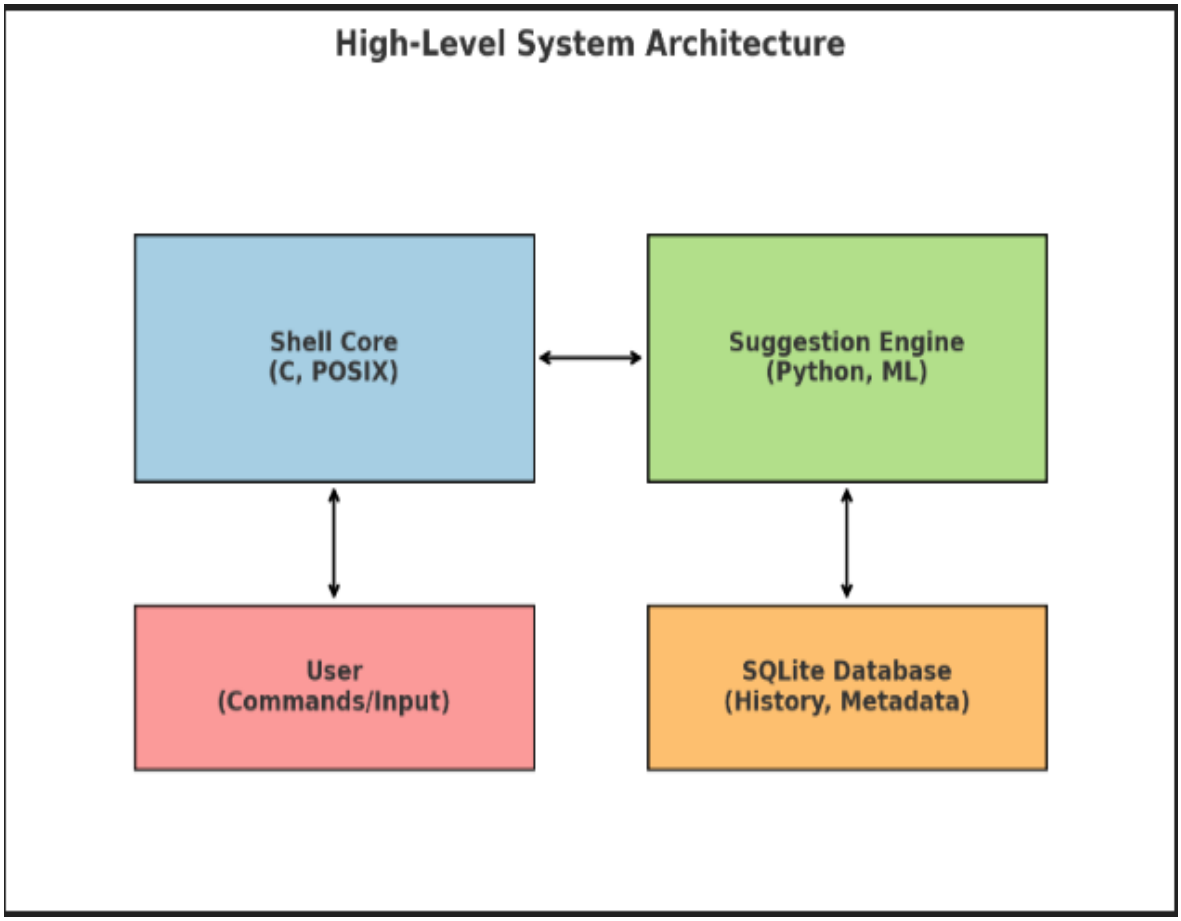
- ***Development & Testing**: Linux (Ubuntu/Arch), macOS (with minor tweaks).*
- ***Tools**: gcc/clang, make, valgrind, gdb, bats for shell testing; Python pytest for ML engine testing.*
- ***Databases & Libraries**: SQLite3, rapidfuzz, numpy, optional scikit-learn.*

System Architecture (High Level Diagram)(2 pts)

(Provide an overview of the system, identifying its main components and interfaces in the form of a diagram using a tool of your choice).

It shows:

- ***Shell Core (C, POSIX)** handling user input, parsing, execution, and communication.*
- ***Suggestion Engine (Python, ML)** providing intelligent recommendations.*
- ***SQLite Database** storing history and metadata.*
- ***User Interface** (commands entered by the user).*
- *Communication via **bidirectional arrows** (Shell ↔ User, Shell ↔ Suggestion Engine, Suggestion Engine ↔ SQLite).*



Project Outcome / Deliverables (1 pts)

(Describe what are the outcomes / deliverables of the project. Max 200 words).

Key Deliverables include:

1. ***Shell Core (C, POSIX)*** *– A custom-built shell supporting command parsing, process management, job control, pipes, redirection, and built-in commands.*
2. ***History Management System*** *– A SQLite-backed store capturing command history with timestamps, working directory, exit status, and tags.*
3. ***Suggestion Engine (Python, ML)*** *– A self-learning module that provides:*
    - *Typo correction for mistyped commands.*
    - *Next-command prediction based on history and context.*
    - *Flag and template recommendations tailored to user patterns.*
    - *Explainable outputs with confidence scores.*
4. ***Inter-Process Communication (IPC) Framework*** *– Seamless data exchange between the shell core and suggestion engine using Unix domain sockets with JSON.*
5. ***Testing Suite*** *– Automated tests (bats for shell, pytest for ML) ensuring reliability, correctness, and robustness.*
6. ***Documentation & User Guide*** *– Clear technical documentation and usage instructions for developers and end-users.*

The final system will demonstrate how ***systems programming and machine learning can be integrated*** *to create a next-generation command-line interface that enhances usability, improves productivity, and continuously learns from user behavior.*

Assumptions

(Describe the assumptions (if any) you are making to solve the problem. Max 100 words)

We assume that the project will be developed and tested on ***Linux-based systems*** *(Ubuntu/Arch), with minimal adaptations needed for macOS. Users are expected to have basic familiarity with Unix commands and workflows. The suggestion engine relies on* ***local command history*** *stored in SQLite, assuming that sufficient data will be generated over time for meaningful predictions. The system assumes offline usage, with no dependency on cloud services or external APIs.*

References

(Provide a list of resources or references you utilised for the completion of this deliverable. You may provide links).

1. ***POSIX Standard*** *– IEEE Std 1003.1, 2017 Edition.*
   https://pubs.opengroup.org/onlinepubs/9699919799/

2. ***Advanced Programming in the UNIX Environment*** *– W. Richard Stevens, Stephen A. Rago.*
3. ***GNU Readline Library Documentation***
   https://tiswww.case.edu/php/chet/readline/rltop.html

4. ***SQLite Documentation***
   https://www.sqlite.org/docs.html

5. ***RapidFuzz: Fuzzy String Matching in Python***
   https://maxbachmann.github.io/RapidFuzz/

6. ***scikit-learn: Machine Learning in Python*** *(optional, if used)*
   https://scikit-learn.org/stable/

7. ***The Linux Programming Interface*** *– Michael Kerrisk.*
8. ***pytest: Python Testing Framework***
   https://docs.pytest.org/

9. ***The Fuck: Magnificent app which corrects your previous console command*** *(as an existing solution reference)*
   https://github.com/nvbn/thefuck