Approach, Optimizations, and Trade-offs

Approach:

To implement the Braille autocorrect system, I first researched the **Levenshtein distance algorithm**, which measures the minimum number of single-character edits (insertions, deletions, or substitutions) needed to change one word into another. This algorithm is commonly used in spelling correction and fuzzy string matching. To build a strong foundational understanding, I referred to the following resources:

1. *What is Levenshtein Distance?* – YouTube link
2. *Edit Distance Between 2 Strings - LeetCode* – YouTube link
3. *Minimum Edit Distance - Dynamic Programming* – YouTube link
4. *Levenshtein Distance Algorithm + Code* – YouTube link

After understanding the algorithm, I coded the Levenshtein distance function using Python and integrated it into a Flask web application. I initially developed the backend in PyCharm and created a basic user interface using HTML and CSS. Throughout the development process, I used **ChatGPT** as a learning assistant to understand edge cases and refine the algorithm's behavior to suit my project's needs.

The core functionality allows users to input QWERTY-based Braille patterns, which are decoded into English characters. If the decoded word does not match a valid English word, the system uses Levenshtein distance to suggest the closest valid word from the dictionary (provided by the `english-words` Python package).

Optimizations:

- The algorithm calculates edit distance between the decoded word and all valid dictionary entries, selecting the closest match.
- For performance, I limited suggestions to the shortest edit distance found, breaking early once the optimal match is found.
- The dictionary is loaded once during server startup for efficiency.

Trade-offs:

- **Accuracy vs. Popularity**: The system suggests the **closest word** based on character-level similarity rather than word frequency or context. This means the suggested word may not always be the most *commonly used* English word but the one with the least edit distance.

- **Performance**: While the approach is sufficient for small to medium-sized dictionaries, a large dictionary could increase response time due to the brute-force nature of the comparison. Optimization with Trie structures or BK-trees was considered but not implemented due to project scope and timeline.

Goal:

My primary goal was to **develop a working prototype** that bridges Braille input with intelligent English word suggestions to improve accessibility and learning tools for visually impaired users or Braille learners.
