**Auto correction model:**

import numpy as np

class KeyboardAutoCorrector:

def \_\_init\_\_(self, dictionary):

self.dictionary = set(word.lower() for word in dictionary)

def levenshtein\_distance(self, s1, s2):

if len(s1) < len(s2):

return self.levenshtein\_distance(s2, s1)

if len(s2) == 0:

return len(s1)

previous\_row = np.arange(len(s2) + 1)

for i, char1 in enumerate(s1):

current\_row = np.zeros(len(s2) + 1, dtype=int)

current\_row[0] = i + 1

for j, char2 in enumerate(s2):

insertions = previous\_row[j + 1] + 1

deletions = current\_row[j] + 1

substitutions = previous\_row[j] + (char1 != char2)

current\_row[j + 1] = min(insertions, deletions, substitutions)

previous\_row = current\_row

return previous\_row[-1]

def auto\_correct(self, word):

word\_lower = word.lower()

if word\_lower in self.dictionary:

return word # The word is already in the dictionary

corrected\_word = self.find\_closest\_word(word\_lower)

return corrected\_word

def find\_closest\_word(self, word):

min\_distance = float('inf')

closest\_word = word

for dictionary\_word in self.dictionary:

distance = self.levenshtein\_distance(word, dictionary\_word)

if distance < min\_distance:

min\_distance = distance

closest\_word = dictionary\_word

return closest\_word

if \_\_name\_\_ == "\_\_main\_\_":

# Example usage with a small dictionary

word\_dictionary = ["python", "programming", "language", "spell", "correction"]

auto\_corrector = KeyboardAutoCorrector(word\_dictionary)

while True:

user\_input = input("Enter a word (type 'exit' to quit): ")

if user\_input.lower() == 'exit':

break

corrected\_word = auto\_corrector.auto\_correct(user\_input)

print(f"Corrected word: {corrected\_word}")