A Simple Spelling Error Corrector with a Transpose-First Heuristic

Uraz Kağan Güneş
Information Retrieval and Search Engines (Fall 2025/26)
Özyeğin University
Email: uraz.gunes@ozu.edu.tr

Abstract

We implement a simple spelling error corrector following the classic edits-at Damerau–Levenshtein-distance-1 paradigm: generate all candidates with a single insertion, deletion, substitution, or adjacent transposition, then pick the highest-frequency candidate from a dictionary built from corpus.txt. As an extension, we propose a lightweight Transpose-First heuristic that prioritizes adjacent transposition candidates when available. On the provided 384-word test set, the baseline attains 0.7370 accuracy. The Transpose-First extension matches the baseline (0.7370), while a First-Letter locking heuristic yields a small decrease (0.7344). We provide a minimal, reproducible C# implementation without external libraries.

1 Introduction

Spelling error correction (SEC) aims to map a misspelled token to its intended form. In the absence of context, a standard approach is to restrict candidates to edits within a Damerau–Levenshtein (DL) distance of 1 and select the most frequent candidate under a unigram model learned from a large corpus. This project implements such a baseline and evaluates a simple extension, as required by the assignment.

2 Method

Dictionary construction. We tokenize corpus.txt with the regex [A-Za-z]+, perform case-folding (ToLowerInvariant), and count word frequencies to obtain a unigram dictionary f(w). Non-alphabetic characters are ignored.

Baseline inference. Given a misspelled token x, we enumerate the set E1(x) of all strings reachable from x by one of four DL operations: insertion, deletion, substitution, or adjacent transposition. We intersect with the dictionary vocabulary V and select the candidate with the maximum frequency; ties are broken deterministically by lexicographic order. If no valid candidate exists, we output a blank line.

Extension: Transpose-First. We first form the set T(x) of candidates produced only by a single adjacent transposition. If $T(x) \cap V \neq \emptyset$, we restrict selection to that set; otherwise, we fall back to the baseline pool $E1(x) \cap V$. This adds no asymptotic cost and preserves the baseline when no transposition candidate is in-vocabulary.

Ablation: First-Letter Lock. For comparison, we tested a heuristic that retains only candidates whose first letter matches that of the misspelling; this can hurt when the initial character is corrupted or when boundary edits occur.

3 Experimental Setup

Data. The dictionary is built from the provided corpus.txt (concatenated public-domain texts and frequency lists). The evaluation uses test-words-misspelled.txt (384 items) and their gold forms in test-words-correct.txt.

Implementation. The system is implemented in C# (.NET 8) without external libraries. Candidate generation enumerates DL=1 operations; selection uses frequency and lexicographic tie-breaking. A CLI exposes predict, eval, and compare commands.

Metric. We report accuracy on the 384-word set, as specified in the assignment.

4 Results

Table 1 reports accuracies. The Transpose-First extension matches the baseline, while First-Letter slightly underperforms.

System	Correct / Total	Accuracy
Baseline (edits1, freq)	283 / 384	0.7370
Extension: First-Letter	282 / 384	0.7344
Extension: Transpose-First	285 / 384	0.7422

5 Analysis

First-Letter can discard correct answers when the initial character is corrupted or when insertion/deletion at the beginning shifts characters. Transpose-First helps on canonical transposition errors and is otherwise neutral, yielding parity with the baseline on this set. Most remaining errors involve candidates outside the dictionary or ties where the gold form is less frequent than a competing in-vocabulary variant.

6 Classes and Responsibilities

Program (Program.cs): CLI entry point. Routes commands (predict, eval, compare). Key methods: Main (argument dispatch), RunPredict (batch predictions to file), RunEval (accuracy computation; optional --out dump), RunCompare (baseline vs extension), Require (option guard), PrintUsage (help). Handles I/O only; no model logic.

ArgParser (ArgParser.cs): Minimal parser for --key value pairs; flags without values default to true. Returns a dictionary used by Program.

CorpusLoader (CorpusLoader.cs): Builds the word→frequency dictionary from corpus.txt using regex [A-Za-z]+ and ToLowerInvariant(). Outputs Dictionary<string,int>. Complexity linear in corpus size.

SpellingOptions (SpellingCorrector.cs): Holds feature flags controlling behavior: UseFirstLetterLockExtension (first-letter), UseTransposePriorityExtension (transpose-first).

SpellingCorrector (SpellingCorrector.cs): Core correction logic. Fields: _freq (word→frequency), _options. Predict(word): generate edits1; intersect with vocabulary; if transpose-first is on and transposition candidates exist, restrict to them; if first-letter is on and candidates share the first letter, restrict; return highest-frequency candidate (lexicographic tie-break). GenerateEdits1(word): all DL=1 edits (insertion, deletion, substitution, adjacent transposition). GenerateTransposesOnly(word): adjacent transpositions only (for tp). No premature 'return input if in corpus' shortcut.

(Optional) DamerauLevenshtein (DamerauLevenshtein.cs): OSA distance implementation for diagnostics; not required by the baseline selection because we enumerate edits1 explicitly

6 Reproducibility

```
# Baseline
dotnet run -- eval --corpus data/corpus.txt \
  --misspelled data/test-words-misspelled.txt \
  --gold data/test-words-correct.txt

# First-Letter
dotnet run -- eval --corpus data/corpus.txt \
  --misspelled data/test-words-misspelled.txt \
  --gold data/test-words-correct.txt \
  --ext first-letter

# Transpose-First
dotnet run -- eval --corpus data/corpus.txt \
  --misspelled data/test-words-misspelled.txt \
  --gold data/test-words-correct.txt \
  --ext tp

# Dump predictions
dotnet run -- eval ... --out out/preds.txt
dotnet run -- predict --corpus data/corpus.txt --input
data/my_misspelled.txt --output out/predictions.txt
```