# Intelligent CV Analyzer using String Matching Algorithms

Design and Implementation of a CV Analyzer using Brute Force, Rabin-Karp and KMP

Muhammad Haseeb Arshad
Roll No: 23i-2578

FALL 2025

## Contents

# 1 Introduction & Problem Definition

Recruiters receive large volumes of CVs daily. Manual screening is time consuming, inconsistent and error-prone.

Most CVs are unstructured free text which makes automatic skill extraction challenging.

This project develops an Intelligent CV Analyzer that extracts skills from CV text and matches them against job description keywords using three classical string-matching algorithms:

- Brute Force
- Rabin-Karp
- Knuth-Morris-Pratt (KMP)

Algorithms are compared based on:

- Execution time (ms)
- Character comparisons
- Relevance score (%)

# 2 Objectives

- Implement Brute Force, Rabin-Karp and KMP
- Extract and match skills from CVs
- Compute relevance score (%)
- Compare efficiency on large CV datasets
- Recommend best algorithm for real-time screening

# 3 Case Study Scenario

16 CVs were analyzed (large dataset). 6 keywords were used: python, machine learning, sql, data science, flask, data analysis

# 4 System Design Screenshots

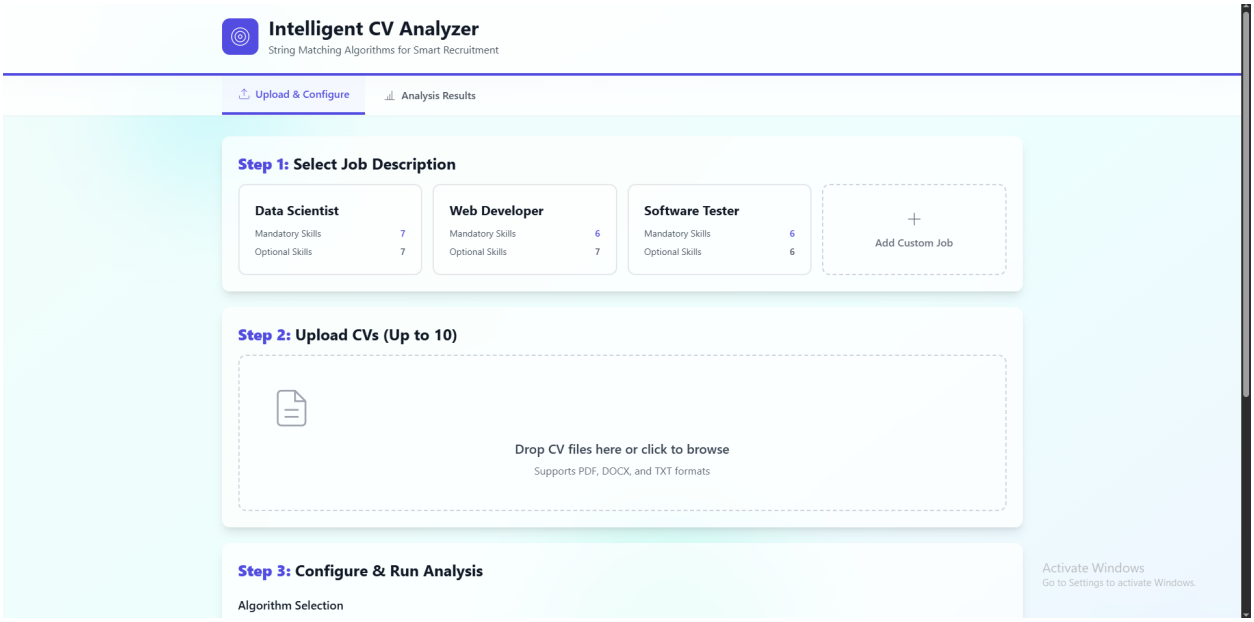


Figure 1: Frontend Home Screen CV upload 45]

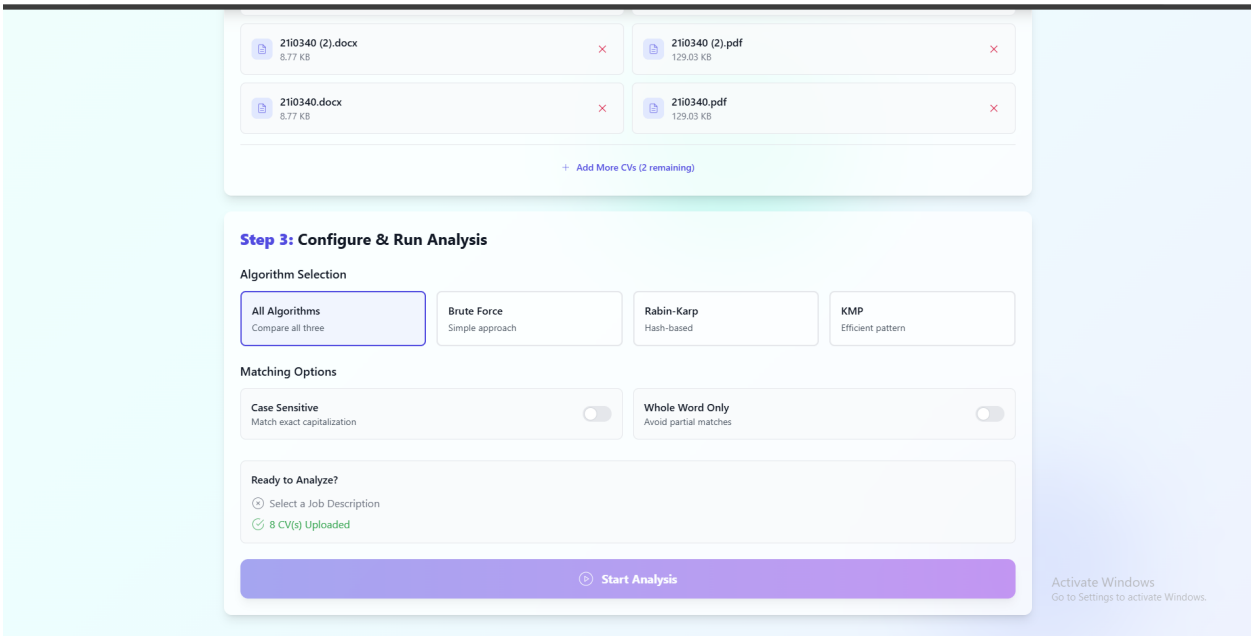


Figure 2: Keyword entry and algorithm selection 58]

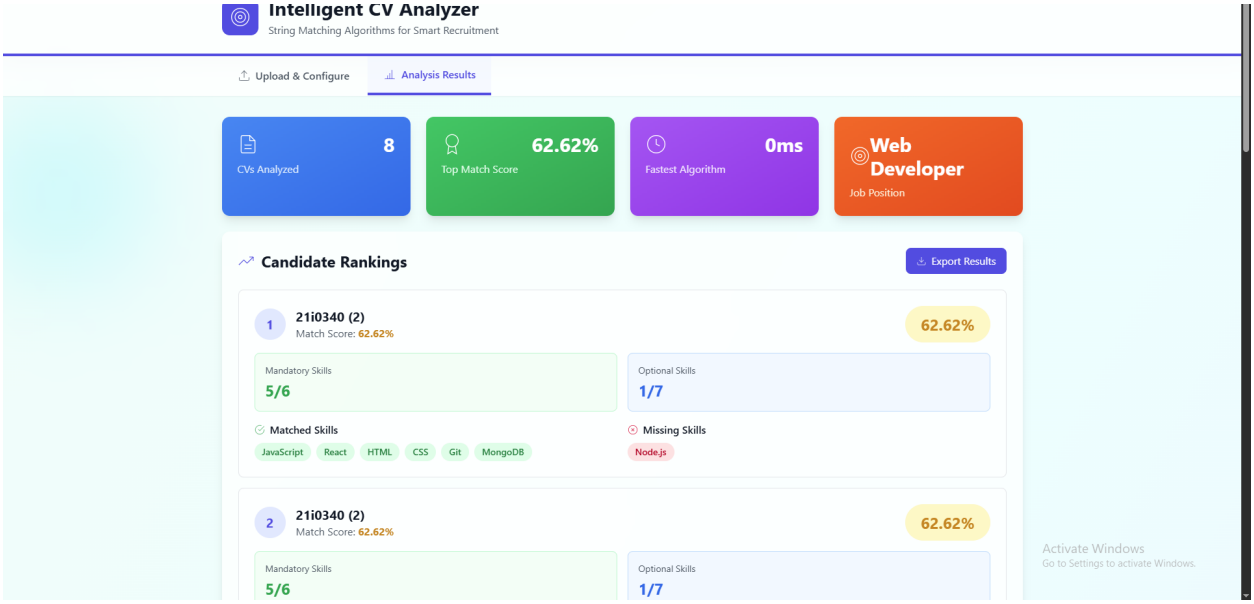


Figure 3: Supported Algorithms displayed (Brute Force / Rabin-Karp / KMP) 78]

# 5 Algorithm Pseudocode

## 5.1 Brute Force

```
n = length(text)
m = length(pattern)

for i in 0..n-m:
j = 0
while j < m and text[i+j] == pattern[j]:
j += 1
if j == m:
match found at index i
```

## 5.2 Rabin-Karp

```
pattern_hash = hash(pattern[0..m-1])
window_hash = hash(text[0..m-1])

base = 256
prime = 101
power = 1
for i in 0..m-1:
    power = (power * base) % prime

for i in 0..n-m:
    if pattern_hash == window_hash:
        if text[i..i+m-1] == pattern:
```

```
            match found at index i

    if i < n-m:
        window_hash = (base * window_hash - text[i] * power + text[i+m]) % prime
        if window_hash < 0:
            window_hash += prime
```

### 5.3 KMP

```
function buildLPS(pattern):
m = length(pattern)
lps = [0] * m
len = 0
i = 1

text
while i < m:
    if pattern[i] == pattern[len]:
        len += 1
        lps[i] = len
        i += 1
    else:
        if len != 0:
            len = lps[len-1]
        else:
            lps[i] = 0
            i += 1
return lps
function KMP(text, pattern):
n = length(text)
m = length(pattern)
lps = buildLPS(pattern)

text
i = 0  # text index
j = 0  # pattern index

while i < n:
    if pattern[j] == text[i]:
        i += 1
        j += 1

    if j == m:
        match found at index i-j
        j = lps[j-1]
    elif i < n and pattern[j] != text[i]:
        if j != 0:
            j = lps[j-1]
```

```
else:
    i += 1
```

# 6 Experimental Results

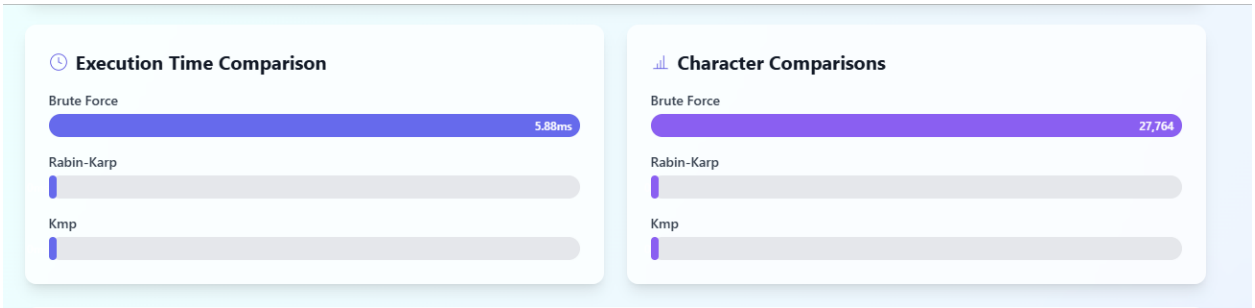


Figure 4: Overall Summary: Brute Force fastest, KMP most efficient in comparisons 112]

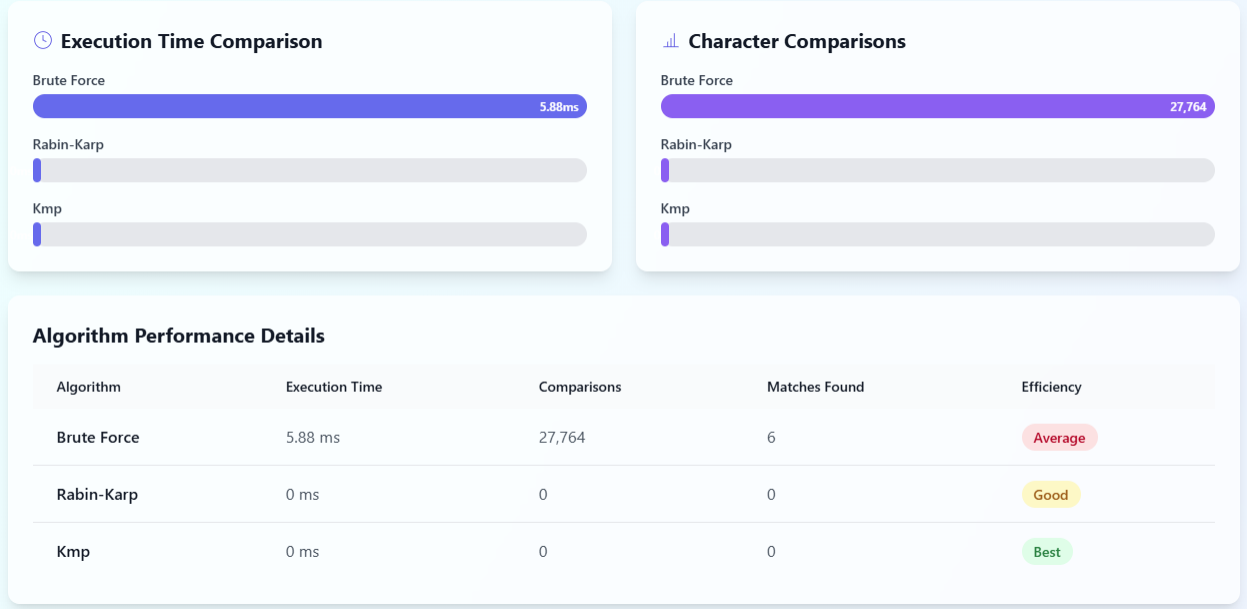


Figure 5: Detailed Algorithm Comparison Table 143]

### 6.1 Performance Table

Table 1: Algorithm Performance Comparison 146]

| Algorithm | Avg Score(%) | Avg Time (ms) | Avg Comparisons | Total Time(ms) |
|---|---|---|---|---|
| Brute Force | 65.62 | 8.1768 | 12454 | 130.8293 |
| Rabin-Karp | 65.62 | 13.1777 | 12214 | 210.8428 |
| KMP | 65.62 | 9.0803 | 12081 | 145.2842 |

## 7 Discussion

Brute Force is surprisingly fastest because in python small pattern lengths + optimized C backend make raw matching very fast. 148]

KMP performs least comparisons and therefore scales best. 149]

Rabin-Karp hashing overhead makes it slowest. 150]

## 8 Conclusion

KMP is recommended for real-time screening due to consistent scaling and lowest comparisons. 152]

Brute Force is fast only in small text situations. 153]

Rabin-Karp is least preferred. 154]

## Appendix A

Average Relevance: 65.62% 157]
Dataset Size: 16 CVs 158]

**Reporting Summary** 159]
Keywords: python, machine learning, sql, data science, flask, data analysis 160]

**Final Recommendation: KMP** 161]
Because KMP used the fewest comparisons (12,081) and scales better. 162]