# Retrieval with Wordle

Kim, Shawn[1], Kleiweg, Martinus[1], and Hernandez, Matthew[2]

[1]Department of Computer Science, University of Arizona
[2]Department of Linguistics, University of Arizona

May 1, 2024

## Abstract

In this paper we describe our end-to-end-implementation of the popular Wordle game, using various Information Retrieval (IR) techniques, together with Reinforcement Learning and ChatGPT. The goal of the game is to guess the word-of-the-day under six attempts, and with the help of feedback in the form of colored tiles. The system operates on algorithms that index five-letter words and perform a boolean search over them. We present analysis over two popular starting words, and make use of an inverted-index to reduce the search after each guess.

**Keywords**
Boolean Retrieval, Reinforcement Learning, Zipf's Law, Inverse Document Frequency, Large Language Models

## 1 Introduction

Wordle is a web-based word game where the player operates on six attempts to guess a five-letter word. In this paper, we propose the idea of efficiently guessing the word-of-the-day by indexing a collection of 14,855 documents and traversing the inverted-index via a tailored intersection algorithm[1] Each guess provides contextual information about the state of the game, which are represented by buckets of objects—the color of the tiles—during the indexing phase.

The search is initialized by the starting word that queries the index and returns a configuration of the game. Our scoring function then recalculates the next best word, and queries the reduced search space until convergence. We aim to have comparable results, if not a small improvement, to that of a human player [1].

[1]Valid guesses are available from the source code. or from this GitHub Repo.

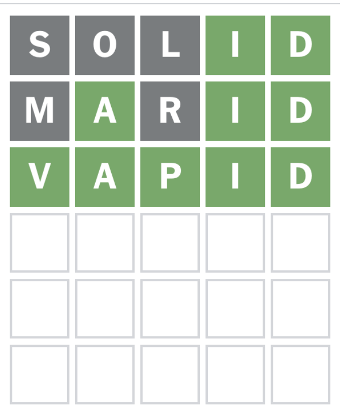

Figure 1: Colored tiles provide feedback for the player.

The paper is organized as follows: in Section 2, we introduce the core implementation of the IR system and describe how the positions of each colored tiles are encoded and describe our approach to retrieving the best document. In Section 3, we measure the performance of our system by the number of guesses made within a game. In Section 4, we discuss the error analysis on our best system and group the errors we observed. Lastly, in Section 5, we attempt to improve our implementation.

## 2 Playing Wordle

We provide a brief survey to the Wordle puzzle, created and developed by software engineer John Wardle, and released in October 2021. In Wordle, the player is allotted six tries to guess a five-letter word, with feedback given in the form of colored boxes indicating whether a) the letter is present and b) whether the letter holds the correct position, see Fig. 1.

The word-of-the-day is chosen from a hand-picked list, according to some criteria specified by Wardle and his wife [2]. A list of 13-14k valid words provides the basis of Wordle; furthermore, the handpicked list is chosen from a "mystery list" as a list of less than 2,309 words.

## 3 Core Implementation

The process of guessing the word-of-the-day can be framed as an Information Retrieval (IR) task, where each five-letter word is a document in a collection and the letters are the atomic units that define the vocabulary.

### 3.1 Indexing

The core implementation of the IR system uses an inverted-index that distinguishes the position of each letter of the alphabet and stores the total count of words in that position, increasing the vocabulary from 26 to 130 terms, see Figure 2. The code is implemented from scratch and provides functionality to view the vocabulary and index.

The terms require no preprocessing and there is no information loss during any stage of the implementation. Each word in indexed in a postings list in alphabetical order.

| **Dictionary** | **Postings List** | |
|---|---|---|
| $a_1$ : 868 | $\longrightarrow$*aahed* | $\longrightarrow$*aalii* |
| $a_2$ : 2682 | $\longrightarrow$.... | $\longrightarrow$*haiku* |
| $a_3$ : 1374 | $\longrightarrow$*abaca* | $\longrightarrow$*abase* |
| ... | $\longrightarrow$..... | $\longrightarrow$..... |
| $f_4$ : 258 | $\longrightarrow$*alefs* | $\longrightarrow$*aleft* |
| $f_5$ : 95 | $\longrightarrow$*aloof* | $\longrightarrow$*blaff* |
| $g_1$ : 685 | $\longrightarrow$*gabba* | $\longrightarrow$*gabby* |
| $g_2$ : 94 | $\longrightarrow$*again* | $\longrightarrow$*agals* |
| ... | $\longrightarrow$..... | $\longrightarrow$..... |
| $z_5$ : 41 | $\longrightarrow$*abuzz* | $\longrightarrow$*afizz* |

Figure 2: Inverted-Index that distinguishes the position and stores their total count.

### 3.2 Boolean Retrieval

The Boolean retrieval model is an appropriate and effective for playing by the mechanics of the word game. We will now describe the composite algorithm that performs various boolean operations and is tailored to handle the response given after each guess.

Three boolean operations (i.e., union, minus, and intersection) form the basis of intersection algorithm—which behaves conceptually similar to the intersection algorithm of two postings list—but the refinement comes from grouping the documents into three sub-lists from their feedback into a response and performs these operations to produce an intersection. The algorithm requires additional arguments, namely the guess, response, index, and the list of valid guesses (or the intersected list).

## 4 Performance

The performance of our IR system is measured by the number of guesses made within a single game. We build up our approach by starting with a naive implementation that scans the list of valid guesses until the WOTD is found. This greedy approach has an unlimited number of tries and will always converge. Next we will introduce a scoring function that potentially reduces the number of tries and limits them to six attempts.

### 4.1 Naive Implementation

A baseline was incorporated to evaluate how well the inverted-index and algorithm were working during the initial stages of the experiment and provided a basis for the improved implementation. To begin, a naive "player" is initialized with a random starting guess and continues intersecting the list every turn until the solution is found. We have chosen two words, *adieu* and *slate*, for comparison[2].

We observed for these two words that the total time to solve 14,855 words was 159 min for *adieu* and 115 min for *slate*, respectively[3]. Moreover, this approach, solves the game alphabetically and is unbounded; the next subsection will introduce as function that is able to recompute the scoring based on the current intersected list in an effort to reduce the number of steps to guess a word.

### 4.2 Scoring Function

A scoring function was introduced to improve upon the naive implementation by returning optimal guesses based on letter distribution either in the collection or the intersected list. The function stores the overall letter frequency of a word and its positional frequency in a hashmap. What follows is generating a list of best words according to their frequencies.

For each letter, a Zipfian scorer computes the penalty for choosing a letter from each posi-

[2]Adieu and Slate are common starting words.

[3]See the benchmarks folder for more information on the baseline.

tion. Frequent letters have a low penalty, and the penalty is incremented with each for loop by adding the previous score. We compute the Zipfian score as,

$$\text{Penalty}_{\text{d, l}} = \frac{1}{\text{Rank}_\text{l} + \beta_\text{l}} \tag{1a}$$

In equation (1a), the penalty is calculated for each letter, $l$ in the distribution, $d$ where $\beta$ is the relative frequency for the letter.

After the penalty is calculated for each position, a function called `calculate_word_scores` creates a hashmap sorted in descending order of their penalty.

An additional hyper-parameter is introduced to prevent words containing a combination of low penalty letters being at the top of the list[4] The hyper-parameter can be decreased during the final implementation if the user believes the solution to contain double characters.

### 4.3 Inverse Document Frequency

The last improvement to the scorer includes the word's inverse document frequency (IDF), from a large collection of documents, in such a way that common words are boosted[5].

We took advantage of the fact that the IDF of an uncommon word is higher—resulting in a larger penalty—and added each word's IDF to its Zipfian score, see Table 1 for a comparison between the two scores.

| **Zipfian Score** | **Zipf+IDF Score** |
|---|---|
| pares: 18.50 | paris: 22.73 |
| tares: 19.20 | pares: 23.49 |
| soare: 19.40 | slate: 23.51 |
| sared: 19.44 | raise: 23.59 |
| bares: 19.50 | mares: 23.86 |

Table 1: Comparison of Top 5 Words

### 4.4 Benchmarks

We will now describe the benchmark used to measure the performance of our best system. The benchmark assigns a system grade based on the average percentage of words successfully solved for each implementation. According to our results on *adieu* and *slate*, we observed that the grade for naive algorithm (System 1) is around 40%, whereas the zipfian scorer (System 2) approaches 60%, see Table 2.

[4]This extra "penalty" rewards heterograms, which are often the best bang for your buck in the game.

[5]An important observation made while playing the game–many solutions are often common words.

| | **System 1** | **System 2** |
|---|---|---|
| *Grade* | 39.96% | 56.64% |
| *Average Tries* | 5.55 | 5.35 |

Table 2: Benchmark between System 1 (Naive) and System 2 (Default).

The overall number of tries between the two systems appears marginal, but the best system on average is most successful in guessing the word-of-the-day within or at six attempts[6]. However, the benchmark is not a true evaluation of the IR system in practice; in the following sections we will include a system analysis on five solutions.

## 5 Error Analysis

The error analysis will be performed for Wordle solutions between the dates April 25—29, 2024, with the starting words *adieu* and *slate*, see Table 3. Our system had stable performance and the two test words behave similarly, also suggesting that the main error could be made during the initialization phase.

| **wotd** / **guess** | intro | vapid | gleam | prune | craft |
|---|---|---|---|---|---|
| adieu | 5 trs | 4 trs | 5 trs | 6 trs | 5 trs |
| slate | 6 trs | 4 trs | 5 trs | 4 trs | 4 trs |

Table 3: Analysis for Wordle solutions between April 25—29, 2024.

Across the provided examples, the Zipfian-IDF implementation had stable performance, in terms of both the number of tries required and the speed of convergence (i.e., the reduced search space). This implementation rapidly converges to the correct solution within a few tries—quite similar to the performance of an average human player. Moreover, this robustness suggests that despite the word's complexity, the implementation typically converges with a strong guess, see Table 4.

## 6 Improved Implementation

We propose two improved implementations to the default system, large language models and reinforcement learning.

[6]Benchmarks are available in the benchmark-system-word.txt file. Further analysis is available in the system analysis.

| | adieu | slate |
|---|---|---|
| guess left | 14,855 | 14,855 |
| guess left | 4,531 | 144 |
| guess left | 492 | 12 |
| guess left | 4 | 3 |
| guess left | 1 | |
| guess left | | |

Table 4: Consistent convergence for the solution *craft*.

## 6.1 ChatGPT Assist

Based on our previous analysis, choosing the right word is crucial to Wordle. One proposed implementation prompted a large language model, ChatGPT-3, previous solutions in an effort to make an informed decision[7]. Unfortunately, introducing ChatGPT prompts doesn't significantly affect the performance or convergence speed compared to the standard implementation.

## 6.2 Reinforcement Learning

As an improved solution, we used reinforcement learning to train a neural network that enables an AI agent to play Wordle. The implementation consists of two main classes.

The `WordleEnv` class manages the game environment by initializing the game state, processing guesses, and providing feedback. This environment simulates reinforcement learning interactions where an agent acts based on the current state and receives feedback in the form of rewards.

The DQNAgent utilizes a neural network that receives inputs of the current game state and a guessed word represented as word indices. This data passes through embedding and dense layers within the network to compute Q-values for each potential action, guiding the agent's decision-making process. Using an epsilon-greedy strategy, the agent balances exploring new guesses and exploiting known rewarding actions. Experience replay is used to update the neural network, allowing the agent to continually refine its strategy based on accumulated knowledge and immediate feedback, to solve Wordle puzzles efficiently.

Surprisingly, the performance between the default system and reinforcement learning system is comparable with both systems totaling 23 tries, see Table 5. We are unable to conclude whether this system is an improvement over the default system and require additional data (i.e., different starting words).

[7]An archive of previous solutions can be found here.

| guess \ wotd | intro | vapid | gleam | prune | craft |
|---|---|---|---|---|---|
| adieu | 5 trs | 5 trs | 5 trs | 5 trs | 5 trs |
| slate | 4 trs | 6 trs | 4 trs | 6 trs | 3 trs |

Table 5: Reinforcement Learning for Wordle solutions between April 25—29, 2024.

## 7 Code

The repository for the code is available on GitHub and contains details on how to reproduce the results in the paper.

In summary, the code is intended to be run in the terminal. There are three main files to run. The default system is the written such that it will be run against **all** words for the selected and print the number of guess in the terminal, see Figure 2.

```
cd src/
python3 benchmark_inv_index_v2.py
Provide your guess: <INSERT-YOUR-WORD>
```

## Concluding Remarks

The implications of our experiment suggest that approaching Wordle as an Information Retrieval task results in comparable performance to a human player[8]. However we have only conducted data for *adieu* and *slate*, and should include a wider variety of starting words. Further improvements can be made to the scoring function, and improve the ChatGPT prompt. Moreover, the comparable performance between reinforcement learning system calls for investigation on both systems.

## Acknowledgements

We want to acknowledge Mihai for providing the frequency counts of a large corpora.

## References

[1] Firecracker. Wordle statistics 2022. https://www.firecrackerpr.com/wordle-statistics-for-2022/#:~:text=The%20US%20is%20ranked%20%2318,a%20national%20average%20of%203.92., Accessed 2024.

[2] Wikipedia, The Free Encyclopedia. Wordle. https://en.wikipedia.org/wiki/Wordle, Accessed 2024.

[8]Our implementation appears to be the first IR approach to Wordle.