Word Search

Problem Statement

Write a HTTP service that provides an endpoint for fuzzy search / autocomplete of English words.

You are given a dataset (`word_search.tsv` - you should have received this via email) that contains 333,333 English words and the frequency of their usage in some corpus. A very small sample is shown below:

```
track 112385243
australia 112197265
discussion 111973466
archive 111971865
once 111882023
others 111397714
entertainment 111394818
agreement 111356320
format 111279626
```

Let us assume we're building a web app where the user types in a **single** word from this list in a search box. We wish to autocomplete the input in the search box.

Your objective is to write a Python app that exposes a single endpoint:

```
GET /search?word=<input>
```

where input is the (partial) word that the user has typed so far. For example, if

the user is looking up `procrastination`, the service might receive this sequence of requests:

`GET /search?word=pro`

`GET /search?word=procr`

`GET /search?word=procra`

and so on.

The response should be a JSON array containing upto 25 results, ranked by some criteria (see below).

Constraints

1. The matching should be fuzzy and tolerate typos. For example, both `grtness` and `graetness` should match `greatness`.
2. Matches can occur anywhere in the string, not just at the beginning. For example, `eryx` should match `archaeopteryx` (among others).
3. The ranking of results should satisfy the following:
    - We assume that the user is typing the beginning of the word. Thus, matches at the start of a word should be ranked higher. For example, for the input `pract`, the result `practical` should be ranked higher than `impractical`.
    - Common words (those with a higher usage count) should rank higher than rare words.
    - Short words should rank higher than long words. For example, given the input `environ`, the result `environment` should rank higher than `environmentalism`.
        - As a corollary to the above, an *exact match* should always be ranked as the first result.
4. The service should be **fast**. You should consider 100ms to be a generous upper bound for the endpoint to return its response (excluding all network latencies). In particular, assume that loading the entire dataset into a database table and using the database engine's text search capabilities is not an acceptable solution.
5. The service can use as much memory as it wants (within reason), but the

upper bound on its memory usage must be knowable in advance.

Submission

Please submit the source code of your solution as a GitHub repo. Extra credit for hosting it in a publicly accessible location for testing.