

# Fuzzy search to correct misspellings and typos

04/08/2020 • 6 minutes to read • 

## In this article

[What is fuzzy search?](#)

[Indexing for fuzzy search](#)

[How to use fuzzy search](#)

[Testing fuzzy search](#)

[See also](#)

Azure Cognitive Search supports fuzzy search, a type of query that compensates for typos and misspelled terms in the input string. It does this by scanning for terms having a similar composition. Expanding search to cover near-matches has the effect of auto-correcting a typo when the discrepancy is just a few misplaced characters.

## What is fuzzy search?

It's an expansion exercise that produces a match on terms having a similar composition. When a fuzzy search is specified, the engine builds a graph (based on [deterministic finite automaton theory](#)) of similarly composed terms, for all whole terms in the query. For example, if your query includes three terms "university of washington", a graph is created for every term in the query `search=university~ of~ washington~` (there is no stop-word removal in fuzzy search, so "of" gets a graph).

The graph consists of up to 50 expansions, or permutations, of each term, capturing both correct and incorrect variants in the process. The engine then returns the topmost relevant matches in the response.

For a term like "university", the graph might have "unversty, universty, university, universe, inverse". Any documents that match on those in the graph are included in results. In contrast with other queries that analyze the text to handle different forms of the same word ("mice" and "mouse"), the comparisons in a fuzzy query are taken at face value without any linguistic

analysis on the text. "Universe" and "inverse", which are semantically different, will match because the syntactic discrepancies are small.

A match succeeds if the discrepancies are limited to two or fewer edits, where an edit is an inserted, deleted, substituted, or transposed character. The string correction algorithm that specifies the differential is the [Damerau-Levenshtein distance](#) metric, described as the "minimum number of operations (insertions, deletions, substitutions, or transpositions of two adjacent characters) required to change one word into the other".

In Azure Cognitive Search:

- Fuzzy query applies to whole terms, but you can support phrases through AND constructions. For example, "Unviersty~ of~ "Wshington~" would match on "University of Washington".
- The default distance of an edit is 2. A value of `~0` signifies no expansion (only the exact term is considered a match), but you could specify `~1` for one degree of difference, or one edit.
- A fuzzy query can expand a term up to 50 additional permutations. This limit is not configurable, but you can effectively reduce the number of expansions by decreasing the edit distance to 1.
- Responses consist of documents containing a relevant match (up to 50).

Collectively, the graphs are submitted as match criteria against tokens in the index. As you can imagine, fuzzy search is inherently slower than other query forms. The size and complexity of your index can determine whether the benefits are enough to offset the latency of the response.

#### ⓘ Note

Because fuzzy search tends to be slow, it might be worthwhile to investigate alternatives such as n-gram indexing, with its progression of short character sequences (two and three character sequences for bigram and trigram tokens). Depending on your language and query surface, n-gram might give you better performance. The trade off is that n-gram indexing is very storage intensive and generates much bigger indexes.

Another alternative, which you could consider if you want to handle just the most egregious cases, would be a **synonym map**. For example, mapping "search" to "serach, serch, sarch", or "retrieve" to "retreive".

## Indexing for fuzzy search

Analyzers are not used during query processing to create an expansion graph, but that doesn't mean analyzers should be ignored in fuzzy search scenarios. After all, analyzers are used during indexing to create tokens against which matching is done, whether the query is free form, filtered search, or a fuzzy search with a graph as input.

Generally, when assigning analyzers on a per-field basis, the decision to fine-tune the analysis chain is based on the primary use case (a filter or full text search) rather than specialized query forms like fuzzy search. For this reason, there is not a specific analyzer recommendation for fuzzy search.

However, if test queries are not producing the matches you expect, you could try varying the indexing analyzer, setting it to a **language analyzer**, to see if you get better results. Some languages, particularly those with vowel mutations, can benefit from the inflection and irregular word forms generated by the Microsoft natural language processors. In some cases, using the right language analyzer can make a difference in whether a term is tokenized in a way that is compatible with the value provided by the user.

## How to use fuzzy search

Fuzzy queries are constructed using the full Lucene query syntax, invoking the [Lucene query parser](#).

1. Set the full Lucene parser on the query (`queryType=full`).
2. Optionally, scope the request to specific fields, using this parameter (`searchFields=<field1,field2>`).
3. Append the tilde (~) operator at the end of the whole term (`search=<string>~`).

Include an optional parameter, a number between 0 and 2 (default), if you want to specify the edit distance (`~1`). For example, "blue~" or "blue~1" would return "blue", "blues", and "glue".

In Azure Cognitive Search, besides the term and distance (maximum of 2), there are no additional parameters to set on the query.

#### ⓘ Note

During query processing, fuzzy queries do not undergo **lexical analysis**. The query input is added directly to the query tree and expanded to create a graph of terms. The only transformation performed is lower casing.

## Testing fuzzy search

For simple testing, we recommend [Search explorer](#) or [Postman](#) for iterating over a query expression. Both tools are interactive, which means you can quickly step through multiple variants of a term and evaluate the responses that come back.

When results are ambiguous, [hit highlighting](#) can help you identify the match in the response.


#### ⓘ Note

The use of hit highlighting to identify fuzzy matches has limitations and only works for basic fuzzy search. If your index has scoring profiles, or if you layer the query with additional syntax, hit highlighting might fail to identify the match.


## Example 1: fuzzy search with the exact term

Assume the following string exists in a "Description" field in a search document: "Test queries with special characters, plus strings for MSFT, SQL and Java."


Start with a fuzzy search on "special" and add hit highlighting to the Description field:

Console	 Copy
<pre>search=special~&amp;highlight=Description</pre>	


In the response, because you added hit highlighting, formatting is applied to "special" as the matching term.

Output	 Copy
<pre>"@search.highlights": {   "Description": [     "Test queries with &lt;em&gt;special&lt;/em&gt; characters, plus strings for MSFT, SQL and Java."   ] }</pre>	


Try the request again, misspelling "special" by taking out several letters ("pe"):

Console	 Copy
<pre>search=scial~&amp;highlight=Description</pre>	


So far, no change to the response. Using the default of 2 degrees distance, removing two characters "pe" from "special" still allows for a successful match on that term.

Output	 Copy
<pre>"@search.highlights": {   "Description": [     "Test queries with &lt;em&gt;special&lt;/em&gt; characters, plus strings for MSFT, SQL and Java."   ] }</pre>	

Trying one more request, further modify the search term by taking out one last character for a total of three deletions (from "special" to "scal"):

Console	 Copy
<pre>search=scal~&amp;highlight=Description</pre>	

Notice that the same response is returned, but now instead of matching on "special", the fuzzy match is on "SQL".

Output	 Copy
<pre>"@search.score": 0.4232868, "@search.highlights": {   "Description": [     "Mix of special characters, plus strings for MSFT, &lt;em&gt;SQL&lt;/em&gt;, 2019, Linux, Java."   ] }</pre>	

The point of this expanded example is to illustrate the clarity that hit highlighting can bring to ambiguous results. In all cases, the same document is returned. Had you relied on document IDs to verify a match, you might have missed the shift from "special" to "SQL".

## See also

- [How full text search works in Azure Cognitive Search \(query parsing architecture\)](#)
- [Search explorer](#)
- [How to query in .NET](#)
- [How to query in REST](#)

---

Is this page helpful?

 Yes  No

