MagiCoder Pro

User Manual

March 5, 2026

Table of Contents

Overview

MagiCoder Pro is a tool that automatically recognizes the terms of a given dictionary within a free form text, such as (but not limited to):

- MedDRA terms in adverse drug reaction reports (ICSRs);
- ICD terms in medical records (EHRs);
- IDMP terms in Summaries of Product Characteristics (SmPCs).

Example outputs are shown in the following table.

Table 1. MagiCoder Pro example outputs.

Source Text	Given Dictionary	Recognized Terms
«After taking the antibiotics, the patient experienced erythema and edema on the wrists and ankles»	MedDRA	«`10015150` Erythema», «`10076565` Wrist edema», «`10002542` Ankle edema»
«Each film-coated tablet contains leflunomide»	IDMP	«`100000085463` Leflunomide», «`100000073665` Film-coated tablet»

Features

It supports multiple dictionaries

MagiCoder Pro has been developed and tested using MedDRA, but it can also be used with other dictionaries. Its algorithm is generic, and using the dictionary you are interested in is simply a matter of loading its terms into the tool's database.

It supports multiple languages

MagiCoder Pro works in Italian, English, Dutch, Spanish, French, and Portuguese, and other languages supported by libstemmer (see the algorithms page).

If the language you need isn't listed, consider pre-processing your source text and translating it into English using an LLM.

It is fast

MagiCoder Pro's algorithm scans each word of the source text once and only once, and returns the results in milliseconds.

As an illustration, on an AWS EC2 t4g.small (2 vCPU, 2 GB RAM) the average latency observed with 8 concurrent users to encode a short text (under 255 characters) is about 21 milliseconds, and throughput is about 1.5 million requests per hour.

It has high reliability

MagiCoder Pro finds up to 90% of the terms in the source text on average (regardless of declensions, punctuation, typos, etc).

A summary of data published in peer-reviewed journal articles is shown in the following table.

Table 2. MagiCoder Pro measured reliability.

Dictionary	Precision	Recall
MedDRA - Italian	91%	86%
MedDRA - English	86%	73%
MedDRA - Dutch	88%	73%

Please remember that:

- *Precision* is the ability to avoid false positives (terms recognized but not present in the text).
- *Recall* is the ability to avoid false negatives (terms not recognized but present in the text).

It is integrable (via API)

MagiCoder Pro is ready to be integrated in other software or workflows through its API.

For more information, please refer to API reference section.

It is self-hostable

MagiCoder Pro is ready to be self-hosted in your infrastructure.

For more information, please refer to self-hosting section.

How it works

Basic encoding algorithm

The main idea of MagiCoder Pro's algorithm can be summarized in five steps:

1. **tokenize the source text**, that is, split it into words (e.g. `["patient", "experienced", "erythema", "and", "edema", "on", "the", "wrists", "and", "ankles"]`);
2. **remove the "stop words"**, which are the non-meaningful ones (e.g. `["patient", "experienced", "erythema", "edema", "wrists", "ankles"]`);
3. **reduce the other words to their stems**, which are the core parts that remain unchanged during inflection, as we want to match the dictionary terms regardless of declensions such as masculine/feminine, singular/plural, and so on (e.g. `["patient", "experienced", "erythema","edema", "wrist", "ankl"]`);
4. **do a single linear scan of stemmed words to vote dictionary terms that contain them** (e.g. if the source text contains `"ankl"` vote the term `10002542 ["ankle", "edema"]`).
5. **order and filter the voted terms by specific criteria and return the best matches!**

With a single linear scan, each word in the source text is read once and only once for voting dictionary terms: this is in contrast to other techniques that read words back and forth and check if any permutation matches some dictionary term (as to match, let's say, the term «10002542 Ankle edema» in «edema of the wrists and ankles» multiple words in different positions have to be taken into account).

This is why MagiCoder Pro is so fast.

To return the best dictionary terms among all the potential (i.e, voted) ones, MagiCoder Pro first sorts them by criteria like:

- coverage (how many voted words does the term have?);
- coverage type (it's a perfect match or a stemmed one?);
- coverage distance (how much the voted term is different from voting words?);
- coverage density (how close are the voting words to each other?);

and then selects them in order until all voting words have been covered, skipping, if necessary, any already selected term or any term that is a prefix of other ones.

As MagiCoder Pro started as a research project in 2015 at the University of Verona, you can read additional information about its algorithm in peer-reviewed journal articles.

Anyway, please also consider that since 2018, the algorithm's development - and the implementation of this product built around it - has been carried out by a private company, so not all details are public.

Advanced encoding features

Synonyms

In MagiCoder Pro new non-dictionary terms can be included as synonyms of dictionary terms; this obviously makes the software more reliable.

Example outputs of the software configured for using synonyms are shown in the following table.

Table 3. MagiCoder Pro example outputs when configured to use synonyms.

Source Text	Given Dictionary	Synonyms	Recognized Terms
«terrible pain»	MedDRA	«`10052412` Terrible pain»	«`10052412` Lancinating pain»
«febrile episodes»	MedDRA	«`10016558` Febrile episodes»	«`10016558` Fever»

If you are using MagiCoder Pro self-hosted, you can include new synonyms by adding them to the `term_synonyms` database table.

Fuzzy matching

In MagiCoder Pro source text and dictionary term words can be compared with an adjustable fuzzy matching: this makes the software insensitive to typos.

Example outputs of the software configured for using fuzzy matching are shown in the following table.

Table 4. MagiCoder Pro example outputs when configured to use fuzzy-matching.

Source Text	Given Dictionary	Recognized Terms
«eadache»	MedDRA	«`10019211` Headache»
«bipokar disorder»	MedDRA	«`10057667` Bipolar disorder»

MagiCoder Pro's fuzzy matching is optional and uses the Damerau-Levenshtein_distance. Note that the greater the allowed distance between source text and dictionary term words, the more likely you are to get some false positives. A more precise (but potentially slower) alternative is to correct typos by pre-processing the source text with an LLM. The best choice depends on your use case.

Parts of Speech

In MagiCoder Pro different source text parts can exclusively vote for different terms (this is based on punctuation, with adjustable punctuation weights).

Example outputs of the software configured for using parts-of-speech are shown in the following table.

Table 5. MagiCoder Pro example outputs when configured to use parts-of-speech. Note that «110066549 Chronic anxiety» is not among the recognized terms.

Source Text	Given Dictionary	Recognized Terms
«chronic pain; anxiety»	MedDRA	«10049475 Chronic pain», «10002855 Anxiety»

Exclusion of dictionary terms

In MagiCoder Pro, some dictionary terms can be excluded if you are not interested in recognizing them: for example, it's useful to exclude MedDRA terms related to investigations when the software is used to encode adverse drug reactions.

Example outputs of the software configured for excluding dictionary terms are shown in the following table.

Table 6. MagiCoder Pro example outputs when configured for excluding dictionary terms. Note that «010045434 Ultrasound scan» is not among the recognized terms.

Source Text	Given Dictionary	Recognized Terms
«ultrasound scan found cyst»	MedDRA	«10011732 Cyst»

If you are using MagiCoder Pro self-hosted, you can mark dictionary terms to avoid by adding them to the `avoid_terms` database table.

Scoped dictionary terms

In MagiCoder Pro, outputs can be dynamically restricted to a subset of dictionary terms, organizing them in lists.

Example outputs of the software configured for scoped dictionary terms are shown in the following table.

Table 7. MagiCoder Pro example outputs when configured for scoped dictionary terms.

Source Text	Given Dictionary	List	Recognized Terms
«Each film-coated tablet contains leflunomide»	IDMP	Pharmaceutical dose forms	«100000073665 Film-coated tablet»
«Each film-coated tablet contains leflunomide»	IDMP	Substances	«100000085463 Leflunomide»

How to use

General concepts

MagiCoder Pro is available in two versions:

- the Managed version is for those who want to reduce operational overhead and get faster access to its features;
- the Self-hosted version is for those who prefer full control over the software and prioritize confidentiality.

In both cases, users will use one or more instances of MagiCoder Pro, each published at different URLs (see Service endpoints).

A single instance manages one dictionary and one language; to encode source text using, for example, MedDRA in Italian and MedDRA in English, you must use two separate MagiCoder Pro instances.

Managed version

In the managed version, **we** are responsible for running MagiCoder Pro.

We provide the hardware (a dedicated virtual machine sized to your needs) and handle the installation and configuration of the software with your preferred dictionaries. On that note, we already have ETL workflows ready for loading MedDRA and IDMP dictionaries; and can implement others upon request.

After our setup, you will receive URLs like:

- https://meddra-it.your-organization.magicoderpro.com
- https://meddra-en.your-organization.magicoderpro.com

and your *authorization key*, and you will be ready to use the software.

Self-hosted version

In the self-hosted version, **you** are responsible for running MagiCoder Pro.

We will provide a Dockerized release of the software and several scripts to start (or stop) one or more containers on a dedicated virtual machine, within your infrastructure.

For example, provided you set up the necessary dictionary files in `db/datasources/`, with:

```
./start_services.sh your-organization.com meddra it,en,fr
```

you will start four containers:

- a reverse proxy that will listen at these URLs:
  - https://meddra-it.your-organization.com
  - https://meddra-en.your-organization.com
  - https://meddra-de.your-organization.com
- three instances of MagiCoder Pro that will perform the encoding (one for each specified dictionary).

The package we provide will include detailed documentation on managing the software and loading the dictionaries, and the source code for all components—except, of course, the code that implements the encoding algorithm.

API Reference

Through the APIs, MagiCoder Pro allows other software (e.g., web applications, ETL workflows, and so on) to programmatically access its functionalities.

Service endpoints

Depending on which software version you use, the service endpoint may differ. Please refer to the following table.

Table 8. MagiCoder Pro endpoints

Version	Endpoint
Managed dedicated version	<dictionary>.<organization>.magicoderpro.com
Self-hosted version	<dictionary>.magicoderpro.<your.domain>

Service authentication

All requests to MagiCoder Pro should contain an authorization header with your *authorization key*.

```
curl -H "AUTHORIZATION_KEY: <key>" "https://<endpoint>/<command>"
```

Service commands

Get an API key

Call `GET http(s)://<endpoint>/generate_api_key?username=<username>` to generate (or change) an API key for the user identified by *username* (**please note that only `admin` user is authorized to call this command**).

Get the dictionary metadata

Call `GET http(s)://<endpoint>/metadata` to get information about the dictionary the web service is using (e.g. name, version, language, etc).

Get an encoding

Call `GET http(s)://<endpoint>/encode?to_encode=<text>` to encode the *text* passed as parameter using the dictionary terms.

You can also use the `POST` method, if you have a long text, or pass other parameters (see below) to modify the algorithm behaviour.

Furthermore, please note that *text* should be [URL encoded](https://en.wikipedia.org/wiki/Percent-encoding) (e.g., `curl -H "AUTHORIZATION_KEY: <key>" --data-urlencode "to_encode=<text>" "http(s)://<endpoint>/encode"`).

Encoding parameters

The API command `encode`, in addition to the mandatory parameter `to_encode`, also has optional ones:

- `avoid`: when *true* (the default) excludes meaningless terms from the response (using a predefined list that for MedDRA could include, for example, all terms beloging to the "Investigations" SOC).
- `allowed_list_codes`: limits the returnable terms to those included in the indicated lists - where a list is a predefined subset of the dictionary terms. Can be repeated (e.g., you can use multiple `allowed_list_codes[]=<list_code>` in your CURL request).
- `coverage_threshold`: when *0.0* (the default) requires all the words in the terms to be voted by the words in the text. You can change this value to *0.5* to require half of them, or *1.0* to require none of them, and so on.
- `distance_from_voters_threshold`: when *0.0* it means that the voters words are required to be identical to term words. Default is *0.5*.
- `stemming`: when *true* (the default) the stemming algorithm is applied, reducing each word to its base form. This makes the encoding insensitive to noun declensions (e.g., «edema of the wrists» → `Wrist edema`).
- `softness`: is how relative order of words in text and terms is managed. When *none* is not considered at all. When *soft* (the default) is considered only when a token votes for more than one term (in this case, the voter is removed from those terms where it has been found in an order different from the one in the term). When *strong* all terms which voters in text are not in the same order they appear in the term are removed.
- `fuzzy_matching`: when *true* (the default) the fuzzy_matching algorithm is applied. This makes the encoding insensitive to typos (e.g., «bipokar disorder» → `Bipolar disorder`).
- `max_word_typos`: set maximum number of word typos allowed.
- `parent`: when *true* (the default) at most one term for each parent term is returned in the response.
- `pos`: when *true* (the default) the part of speech algorithm is applied. This makes the encoding take punctuation into account (e.g., «chronic pain; anxiety» → `Chronic pain` and `Anxiety`, and not `Chronic anxiety`).
- `punctuation_weight`: set the weight (a positive integer or "infinity") associated to each punctuation mark, through a JSON hash (e.g., using something like `--data-urlencode "punctuation_weight={\";\": \"1\"}"` in your CURL request).
- `voters_additional_info`: when *true* voters words and ranges are returned for each term, in the response.

Encoding response

A generic request like:

```
curl -H "AUTHORIZATION_KEY: <key>" --data-urlencode "to_encode=mal di testa"
```

```
"http://<endpoint>/encode?voters_additional_info=true"
```

will usually have a JSON response like:


```
[
  {
    "id": "10019211",
    "name": "Cefalea",
    "name_words": [
      "cefalea"
    ],
    "name_word_stems": [
      "cefale"
    ],
    "voters": [
      0,
      4
    ],
    "voted": [
      0,
      1
    ],
    "coverage": 0,
    "voters_positioning_in_text": 1,
    "voters_positioning_in_term": 1,
    "stemming_usage": 0,
    "distance_from_voters": 0,
    "fuzzy_matching_usage": 0,
    "avoided": false,
    "parent": "10019211",
    "synonym_usage": 1,
    "synonym_id": "242",
    "synonym": "mal di testa",
    "voters_additional_info": {
      "tokenization": [
        "mal",
        " ",
        "di",
        " ",
        "testa"
      ],
      "words": "mal testa",
      "range": "0 3;7 12"
    }
  }
]
```


structured as follows:

- `id` (String) is the ID of the retrieved term.
- `name` (String) is the name retrieved term.
- `name_words` (Array<String\>) are the words composing the retrieved term.
- `name_word_stems` (Array<String\>) are the *stems* of the words composing the retrieved term.
- `voters` (Array<Integer\>) are the indexes of the words in the original free text that have been associated with this term. Words count starts from **0** and counts also for whitespaces.
- `voted` (Array<Integer\>) are the indexes of the words in the term that have been voted by voters. Words count starts from **0** and counts also for whitespaces.
- `stemming_usage` (Boolean) is valued to *1* if the term has been retrieved (also) through the stem of one of its words, **0** otherwise.
- `distance_from_voters` (Float) is the syntactic "edit" distance (actually Pair Distance is used) between term and corresponding words found in the input text (i.e, between *voters* and *voted*).
- `coverage` (Float) is measured in *[0,1]* and represents the percentage of term's words found in the original free text. **0** represents complete coverage (all words of the term have been found), while **1** represents empty coverage.
- `voters_positioning_in_text` (Float) measure expressing how much voters are close one to each other in the input text (i.e., the "density" of the voters in the text). Low values mean voters are close and are preferable.
- `voters_positioning_in_term` (Float) is measured in **[0, +inf]** and expresses how much voted words are close one to each other in the term (i.e., the "density" of the voted words in the term). Low values mean they are close and are preferable.
- `synonym_usage` (Boolean) is valued to *1* if the term has been retrieved through a synonym, *0* otherwise.
- `fuzzy_matching_usage` (Boolean) is valued to *1* if the term has been retrieved through fuzzy matching, *0* otherwise.
- `synonym_id` (String) is the identifier of the synonym used to retrieve this term, when applicable (i.e., synonym_usage is *1*), nil otherwise.
- `synonym` (String) is the synonym used to retrieve this term, when applicable (i.e., *synonym_usage* is **1**), nil otherwise.
- `voters_additional_info` (Hash):
  - `tokenization` is the array of tokens in which the free text has been splitted;
  - `words` is the space-separated list of the voters words;
  - `range` is the semicolon-separated list of ranges. Each range is of the form "**start-offset end-offset**" where start-offset is the index of the first character of the range in the original text, i.e. the number of characters in the document preceding it. The end-offset is the index of the first character after the considered voter. Thus, the character in the end-offset position is not included in the voter span.

In the case the request cannot be successfully completed, the JSON response instead describes the occurred error and is structured as follows:

- `reply_type` (String): the constant string "Error";
- `error_code` (Integer): the error identifier;
- `error_description` (String): a short description of the error.

For example, if no text to encode is provided, the following response will be obtained:

```
{
"reply_type":"error",
"error_code":1,
"error_description":"Missing 'to_encode' parameter."
}
```

Firefox Add-On

Overview

MagiCoder Pro for Firefox is a browser add-on that lets users encode the selected text on any website, or PDF, opened in the browser, using the dictionaries of their choice.

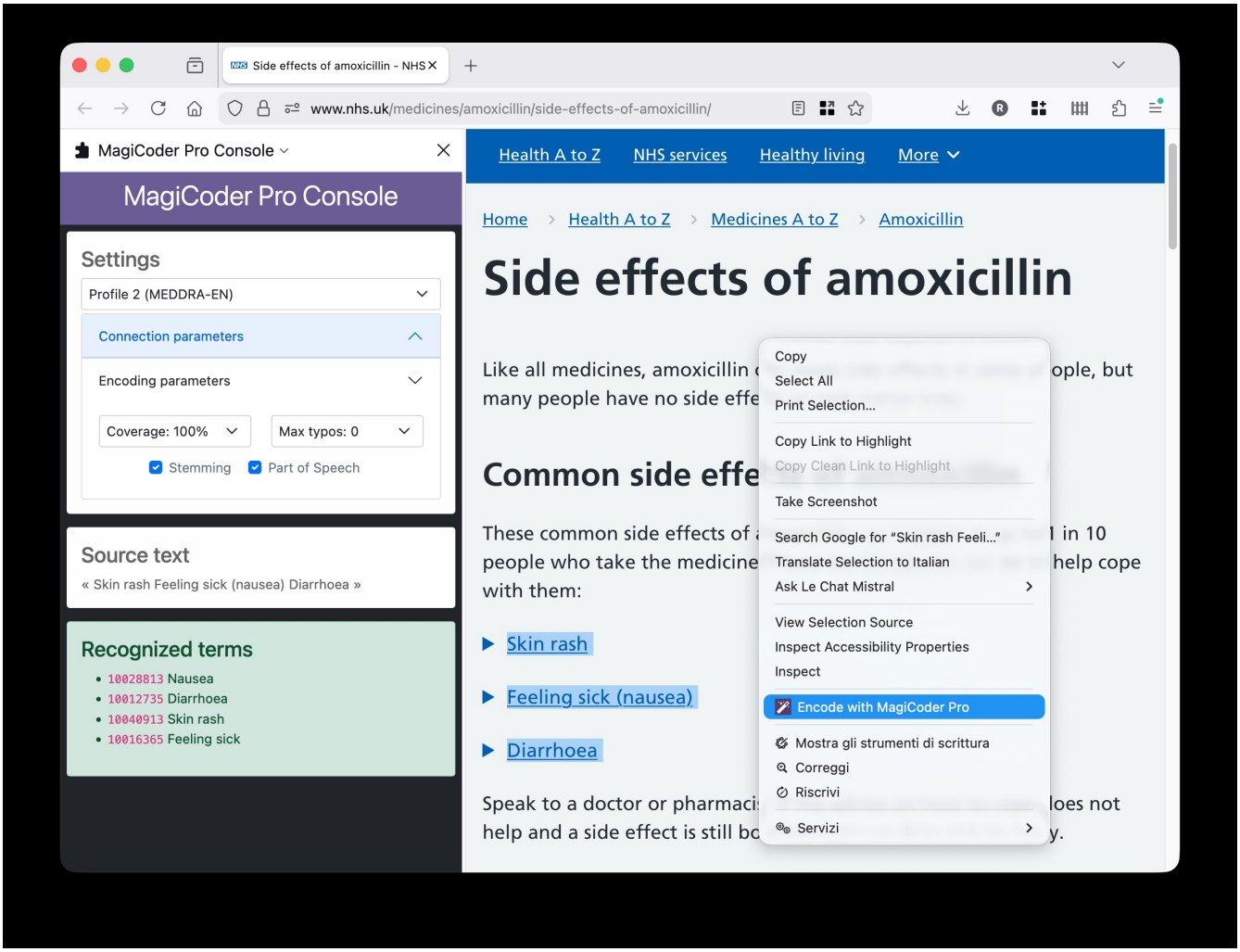

Figure 1. MagiCoder Pro console used to encode in MedDRA an English text.

Installation

The add-on can be installed by clicking the **[Install add-on]** button published on MagiCoder Pro website.

> ⚠ You will be warned that the add-on will access website contents — this is normal, as it needs to encode selected text.

Configuration

To use the add-on, start by opening its sidebar, by selecting the browser menu item **View › Sidebar › MagiCoder Pro Console**. Please note that you can widen it (to see more content at once) by

dragging its right border.

If it's your first time using the add-on, or you want to configure it to use a new MagiCoder Pro instance (tied to a specific dictionary), select a *Profile* (e.g., *Profile 1*) in the *Settings* box and edit the *Connection parameters*. You will be asked to enter the MagiCoder Pro instance *URL*, and your *authorization key*.

Why do you have to enter this information?

That's because the encoding is not performed by the add-on itself, but by some MagiCoder Pro instances running somewhere over the Internet, or on your own infrastructure (see the section "How to Use").

The role of the add-on is to connect your browser with these instances and let you easily encode contents using your preferred dictionaries.

In the *Settings* box you can also adjust *Encoding parameters*:

- *Coverage* controls how many words of a returned dictionary term should be voted by the words in the source text (e.g., 100% means all, 50% half of them);
- *Max typos* controls if fuzzy matching is applied;
- *Stemming* controls if stemming is applied;
- *Parts of Speech* controls if parts of speech are applied.

For more information about *Encoding parameters*, please refer to section How it works.

Usage

1. Select the content you want to encode.
2. Right-click to open the contextual menu.
3. Click the *Encode with MagiCoder Pro* item.
4. Get the recognized terms in the left sidebar.

Appendix A: Peer-Reviewed Articles

- Zorzi M., Combi C., Lora R., Pagliarini M., and Moretti U.. **Automagically Encoding Adverse Drug Reactions in MedDRA**. In 2015 International Conference on Healthcare Informatics (ICHI), pp. 90-99. Dallas, TX, USA, October 21-23, 2015. IEEE Computer Society. DOI: 10.1109/ICHI.2015.18.
- Zorzi M., Combi C., Pozzani G., Moretti U.. **Mapping Free Text into MedDRA by Natural Language Processing: A Modular Approach in Designing and Evaluating Software Extensions**. In ACM-BCB '17 Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics, pp. 27-35. Boston, MA, USA, August 20-23, 2017. ACM. DOI: 10.1145/3107411.3107431.
- Combi C., Zorzi M., Pozzani G., Moretti U., Arzenton E.. **From narrative descriptions to MedDRA: Automagically encoding adverse drug reactions**. Journal of Biomedical Informatics, 84, 184-199. 2018. DOI: 10.1016/j.jbi.2018.07.001.
- Combi C., Zorzi M., Pozzani G., Arzenton E., Moretti U.. **Normalizing spontaneous reports into MedDRA: Some experiments with MagiCoder**. IEEE Journal of Biomedical and Health Informatics, 23(1), 95-102. 2019. DOI: 10.1109/JBHI.2018.2861213.