The Journal of Open Source Software

search-query: An open source Python library for academic search queries

Peter Eckhardt [1], **Katharina Ernst** [1], **Thomas Fleischmann** [1], **Anna Geßler** [1], **Karl Schnickmann** [1], **Laureen Thurner** [1], and **Gerit Wagner** [2]

1 Otto-Friedrich-Universität Bamberg **2** Frankfurt School of Finance & Management

DOI: 10.21105/joss.08775

Software

- Review
- Repository
- Archive

Editor: Claudia Solis-Lemus

Reviewers:

- @amcandio
- @tnehezd

Submitted: 26 June 2025
Published: 19 February 2026

Summary

search-query is an open source Python library for academic literature search queries. Its core capabilities include linting to detect syntactic and logical errors, translation to convert queries across database-specific syntaxes, improvement to programmatically refine queries, and automation to integrate searches into reproducible workflows. The library currently supports *Web of Science*, *PubMed*, and *EBSCOHost*, with parsers validated against a comprehensive set of peer-reviewed queries from *searchRxiv*. Unlike existing proprietary and web-based software, *search-query* offers programmatic access and thereby supports integration into research workflows, contributing to the automation of systematic literature reviews.

Keywords: Python, search query, literature search, literature review, meta-analysis.

Statement of Need

Researchers conducting meta-analyses and other types of literature reviews rely on database searches as the primary search technique (Hiebl, 2023). This involves the design of Boolean queries, the translation for specific databases, and the execution. Requirements include: (1) query translation, i.e., parsing a query in one database specific syntax and serializing it in another syntax (e.g., Al-Zubidy & Carver, 2019; Sturm & Sunyaev, 2019), (2) query linting, i.e., identifying syntactical errors or warning of known database errors (e.g., Li & Rainer, 2023; Singh & Singh, 2017), (3) query improvement, i.e., manipulation of query objects to understand and enhance performance (e.g., Scells et al., 2020), and (4) automation, i.e., offering programmatic or API access to integrate with existing workflows (e.g., Beller et al., 2018; O'Connor et al., 2018).

Overview of search-query capabilities

search-query treats queries as objects rather than static strings. Query objects can be created programmatically or parsed from search strings, giving researchers access to the following capabilities (illustrated in Figure 1):

- **Load:** *search-query* provides parsing capabilities to ingest search queries from both raw strings and JSON files. It parses database-specific query strings into internal, object-oriented representations of nested queries consisting of operators, search terms, and field restrictions. Currently, parsers are available for *Web of Science*, *PubMed*, and *EBSCOHost*. The *load* capability is extensible, and the documentation outlines how to develop parsers for additional databases.
- **Save:** Researchers can serialize the query object back into a standard string or JSON file for reporting and reuse. This facilitates transparency and reproducibility by allowing

search strategies to be easily reported, shared or deposited.

- **Lint:** *search-query* includes linters to detect syntactical errors or inconsistencies that might compromise the search. It can check for issues such as unbalanced parentheses, logical operator misuse, or database-specific syntax errors. The validation rules are based on an analysis of a large corpus of real-world search strategies from the *searchRxiv* registry, revealing that many published queries still contained errors even after peer review. By identifying such problems early, linters can help researchers validate and refine queries before execution. The linting component can be extended to cover more databases and incorporate new messages, such as warnings for database-specific quirks.
- **Translate:** The library can convert a query from one database syntax into another, enabling cross-platform use of search strategies. Using a generic query object as an intermediate representation, *search-query* currently supports translations between *Web of Science*, *PubMed*, and *EBSCOHost*. Such query translation capability can eliminate manual efforts for rewriting queries and reduce the risk of human error during translation. In line with the vision of seamless cross-database literature searches, future development will focus on adding more databases to the translation repertoire.
- **Improve:** Beyond basic syntax checking and translation, *search-query* aims to support query improvement to enhance recall and precision. When queries are represented as manipulable objects, researchers can programmatically experiment with modifications — for example, adding synonyms or adjusting field scopes — to observe how these changes affect the search results. In future work, this improvement capability may be augmented with more automated suggestions and optimizations.
- **Automate:** Automation primarily refers to the integration with systematic review management systems, such as CoLRev (Wagner & Prester, 2025). The library offers programmatic access via its Python API, which means it can be embedded in scripts and pipelines to run searches automatically. It also provides a command-line interface and pre-commit hooks (*Git*), allowing researchers to incorporate query validation into version control and continuous integration setups.

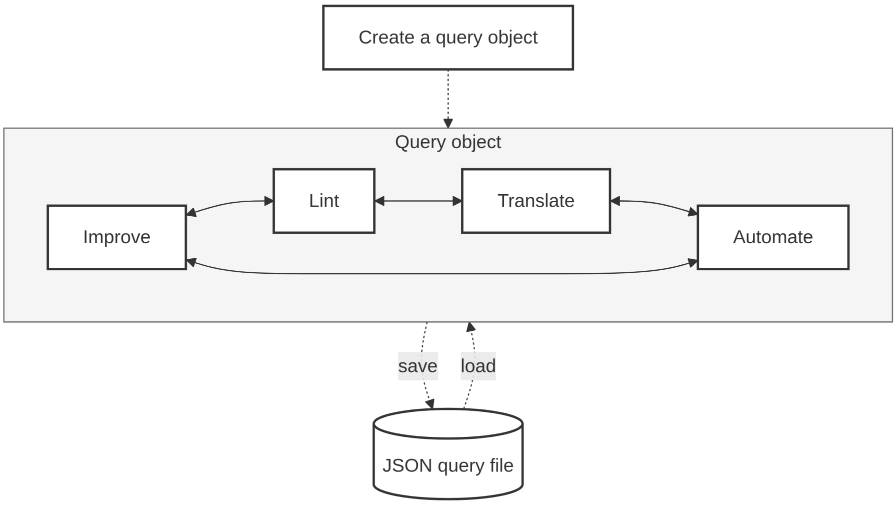


Figure 1: Core capabilities of the *search-query* library

Example usage

Load

```python
# Option 1: Parse query file
from search_query.search_file import load_search_file

search_file = load_search_file("search-file.json")
wos_query = parse(search_file.search_string, platform=search_file.platform)

# Option 2: Parse query string
from search_query.parser import parse

wos_query = parse("digital AND work", platform="wos")

# Option 3: Construct query programmatically
from search_query import OrQuery, AndQuery

digital_synonyms = OrQuery(["digital", "virtual", "online"])
work_synonyms = OrQuery(["work", "labor", "service"])
query = AndQuery([digital_synonyms, work_synonyms], field="title")

# Option 4: Load query from database
from search_query.database import load_query

FT50_journals = load_query("journals_FT50")
```

Save

When saving a query, the JSON format is based on the standard proposed by Haddaway et al. (2022).

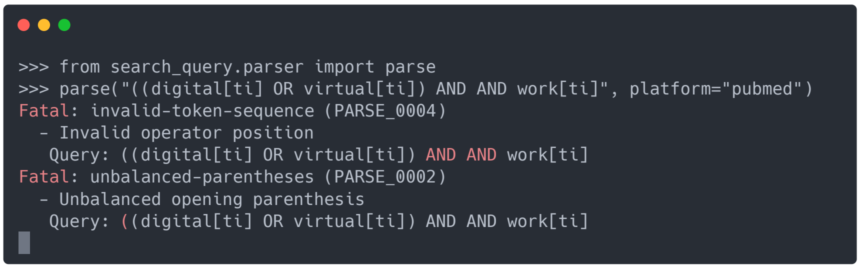
```python
from search_query import SearchFile

search_file = SearchFile(
   query=pubmed_query,
   authors=[{"name": "Gerit Wagner"}],
   record_info={},
   date={}
)

search_file.save("search-file.json")
```

Lint

The parser automatically emits linter messages that identify errors and provide suggestions for improvement. This helps researchers maintain high-quality—even in complex search strategies. In fact, search queries used in literature reviews are often long and complex: Peer-reviewed queries from *searchRxiv*, for instance, average over 2,900 characters and include around 60 parentheses. Such queries are difficult to analyze visually—both for researchers constructing them and for reviewers assessing their validity. The linters identify six categories of errors, which are illustrated briefly in the following.

Parsing errors highlight critical syntax errors that prevent a query from being parsed. Typical examples include unmatched parentheses, misplaced logical operators, or invalid token sequences.

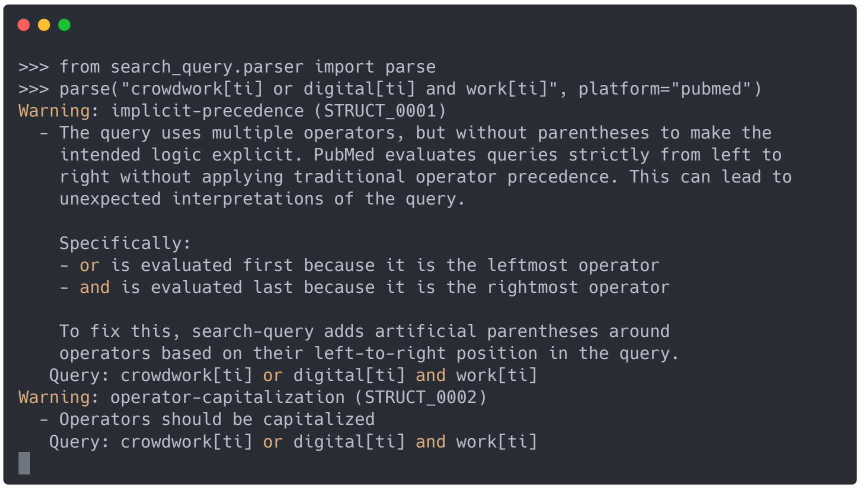
These errors usually require correction before any further processing or database execution is possible.



```
>>> from search_query.parser import parse
>>> parse("((digital[ti] OR virtual[ti]) AND AND work[ti]", platform="pubmed")
Fatal: invalid-token-sequence (PARSE_0004)
  - Invalid operator position
   Query: ((digital[ti] OR virtual[ti]) AND AND work[ti]
Fatal: unbalanced-parentheses (PARSE_0002)
  - Unbalanced opening parenthesis
   Query: ((digital[ti] OR virtual[ti]) AND AND work[ti]
```

Figure 2: Examples of parsing errors

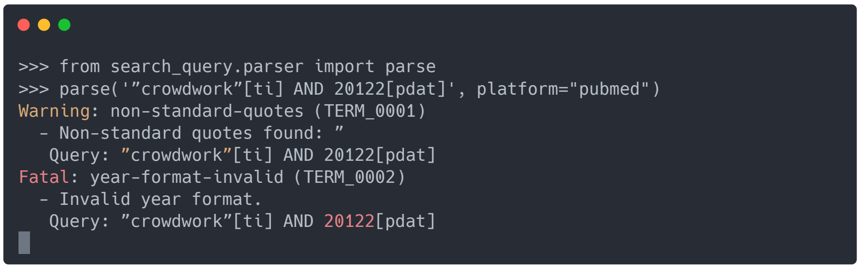
Query structure errors highlight errors that affect the validity or clarity of a query's logical structure, such as implicit operator precedence or inconsistent capitalization of Boolean operators. These warnings help ensure that the intended logic is clear and that the query remains readable and easy to verify.



```
>>> from search_query.parser import parse
>>> parse("crowdwork[ti] or digital[ti] and work[ti]", platform="pubmed")
Warning: implicit-precedence (STRUCT_0001)
  - The query uses multiple operators, but without parentheses to make the
    intended logic explicit. PubMed evaluates queries strictly from left to
    right without applying traditional operator precedence. This can lead to
    unexpected interpretations of the query.

    Specifically:
    - or is evaluated first because it is the leftmost operator
    - and is evaluated last because it is the rightmost operator

    To fix this, search-query adds artificial parentheses around
    operators based on their left-to-right position in the query.
   Query: crowdwork[ti] or digital[ti] and work[ti]
Warning: operator-capitalization (STRUCT_0002)
  - Operators should be capitalized
   Query: crowdwork[ti] or digital[ti] and work[ti]
```

Figure 3: Examples of query structure errors

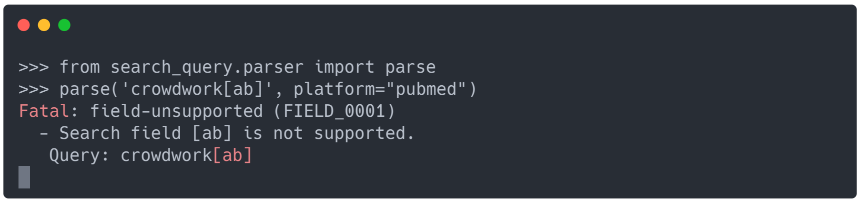
Term errors identify suspicious or malformed search terms, such as non-standard quotes or invalid date formats. These errors may cause databases to interpret the terms incorrectly or fail to return relevant results.



```
>>> from search_query.parser import parse
>>> parse('”crowdwork”[ti] AND 20122[pdat]', platform="pubmed")
Warning: non-standard-quotes (TERM_0001)
  - Non-standard quotes found: ”
   Query: ”crowdwork”[ti] AND 20122[pdat]
Fatal: year-format-invalid (TERM_0002)
  - Invalid year format.
   Query: ”crowdwork”[ti] AND 20122[pdat]
```

Figure 4: Examples of term errors

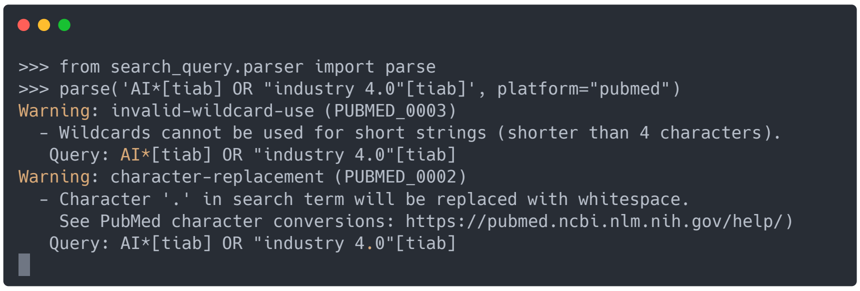
Field errors point to missing, implicit, or unsupported field tags. These errors may lead to incorrect interpretations or cause the query to fail during execution in the database.



```
>>> from search_query.parser import parse
>>> parse('crowdwork[ab]', platform="pubmed")
Fatal: field-unsupported (FIELD_0001)
  - Search field [ab] is not supported.
   Query: crowdwork[ab]
```

Figure 5: Examples of field errors

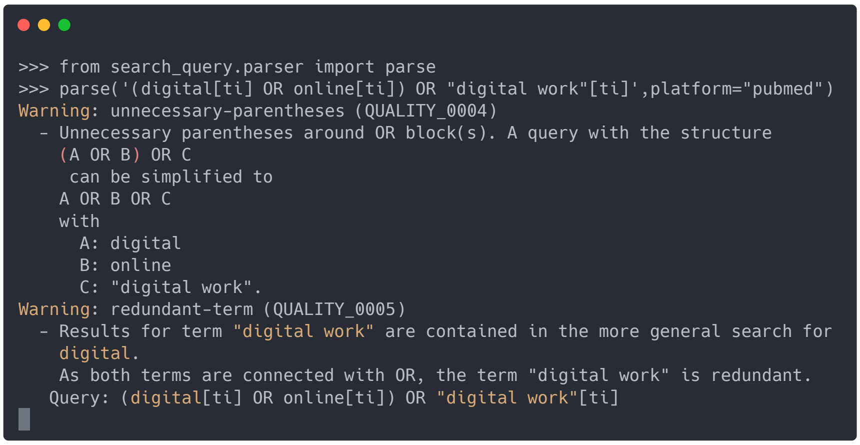
Database errors flag platform-specific quirks and limitations that may not be obvious to users. These include constraints on wildcard usage, invalid characters, and limitations of proximity operators. These errors can cause queries to execute incorrectly or fail, despite appearing syntactically valid.



```
>>> from search_query.parser import parse
>>> parse('AI*[tiab] OR "industry 4.0"[tiab]', platform="pubmed")
Warning: invalid-wildcard-use (PUBMED_0003)
  - Wildcards cannot be used for short strings (shorter than 4 characters).
   Query: AI*[tiab] OR "industry 4.0"[tiab]
Warning: character-replacement (PUBMED_0002)
  - Character '.' in search term will be replaced with whitespace.
    See PubMed character conversions: https://pubmed.ncbi.nlm.nih.gov/help/)
   Query: AI*[tiab] OR "industry 4.0"[tiab]
```

Figure 6: Examples of database errors

Best practice qualities include recommendations for constructing effective search queries. These include alerts about redundant terms, unnecessary parentheses or complex query structures.



```
>>> from search_query.parser import parse
>>> parse('(digital[ti] OR online[ti]) OR "digital work"[ti]',platform="pubmed")
Warning: unnecessary-parentheses (QUALITY_0004)
  - Unnecessary parentheses around OR block(s). A query with the structure
    (A OR B) OR C
     can be simplified to
    A OR B OR C
    with
      A: digital
      B: online
      C: "digital work".
Warning: redundant-term (QUALITY_0005)
  - Results for term "digital work" are contained in the more general search for
    digital.
    As both terms are connected with OR, the term "digital work" is redundant.
   Query: (digital[ti] OR online[ti]) OR "digital work"[ti]
```

Figure 7: Examples of best practice qualities

In addition, it is possible to access linter messages programmatically:

```python
from search_query.linter import lint_file

messages = lint_file(search_file)
```

Translate

```
query_string = '("dHealth"[Title/Abstract]) AND ("privacy"[Title/Abstract])'
pubmed_query = parse(query_string, platform="pubmed")
wos_query = pubmed_query.translate(target_syntax="wos")
print(wos_query.to_string())
# Output:
# (AB="dHealth" OR TI="dHealth") AND (AB="privacy" OR TI="privacy")
```

Capability to improve and automate search queries focuses on the programmatic use of *search-query* for custom logic and use cases (e.g., writing tailored functions). As these features are designed for flexible integration into code-based workflows, it is hard to illustrate them through generic examples; instead, guidance can be found in the online documentation.

Related software packages

Table 1 provides an overview of related software packages and a comparison with *search-query*. The leading query translators, *Polyglot search* and *Litsonar*, are proprietary and delivered through websites, i.e., without integration capabilities. *Polyglot search* supports a more comprehensive selection of databases (15) and initial validation hints (Clark et al., 2020). *Litsonar* supports seven databases, but does not offer parsers (Sturm & Sunyaev, 2019). As software packages available under open source licenses, *Medline Transpose* (Wanner & Baumann, 2019) is a JavaScript-based website that translates queries between three databases, and *litsearchr* (Grames, 2020) is an R library that supports semi-automated generation of queries based on text-mining techniques.

Software package	Setup	License	*Load*	*Save*	*Lint*	*Translate*	*Improve*	*Automate*
Polyglot search	Website	Proprietary	○	●	○	●	-	○
LitSonar	Website	Proprietary	○	●	-	●	-	○
Medline Transpose	Website	MIT	○	●	-	●	-	○
litsearchr	R library	GPL-3	○	●	-	-	●	●
search-query	Python	MIT	●	●	●	●	●	●

Table 1: Overview of related software packages (- no support, ○ limited support, ● support)

In comparison to related software packages, *search-query* provides a Python library, released under the MIT open source license. It is extensible and currently supports three databases (*Web of Science*, *PubMed*, *EBSCOHost*) for query translation and query validation. The query parsers were tested with a comprehensive selection of peer-reviewed queries from *searchRxiv* (White, 2024). Testing showed that a significant number of queries still contained errors after passing the peer-review process, further highlighting the need for linters like *search-query*.

Acknowledgments and outlook

The development of *search-query* originated from a series of student thesis projects at the Digital Work Lab, Otto-Friedrich-Universität Bamberg (Eckhardt, 2025; Ernst, 2024; Fleischmann, 2025; Geßler, 2025; Schnickmann, 2025). Looking forward, we envision *search-query* growing through both community and academic contributions. The developer documentation provides guidance on extending the library, for example, adding support for new databases or custom linter rules. Our goal is to build an open platform and continually expand *search-query*'s capabilities in line with the needs of researchers working on literature review projects.

References

Al-Zubidy, A., & Carver, J. C. (2019). Identification and prioritization of SLR search tool requirements: An SLR and a survey. *Empirical Software Engineering*, *24*, 139–169. https://doi.org/10.1007/s10664-018-9626-5

Beller, E., Clark, J., Tsafnat, G., Adams, C., Diehl, H., Lund, H., Ouzzani, M., Thayer, K., Thomas, J., Turner, T., Xia, J., Robinson, K., & Glasziou, P. (2018). Making progress with the automation of systematic reviews: Principles of the international collaboration for the automation of systematic reviews (ICASR). *Systematic Reviews*, *7*(1). https://doi.org/10.1186/S13643-018-0740-7

Clark, J. M., Glasziou, P., Del Mar, C., Bannach-Brown, A., Stehlik, P., & Scott, A. (2020). *Polyglot search*. https://sr-accelerator.com/#/polygloT

Eckhardt, P. (2025). *Advances in literature searches: Evaluation, analysis, and improvement of web of science queries* [Thesis]. Otto-Friedrich-University of Bamberg.

Ernst, K. (2024). *Towards more efficient literature search: Design of an open source query translator* [Thesis]. Otto-Friedrich-University of Bamberg.

Fleischmann, T. (2025). *Advances in literature search queries: Validation and translation of search strings for EBSCO host* [Thesis]. Otto-Friedrich-University of Bamberg.

Geßler, A. (2025). *Design of an emulator for API-based academic literature searches* [Thesis]. Otto-Friedrich-University of Bamberg.

Grames, E. (2020). *Litsearchr*. https://elizagrames.github.io/litsearchr/

Haddaway, N. R., Rethlefsen, M. L., Davies, M., Glanville, J., Mcgowan, B., Nyhan, K., & Young, S. (2022). A suggested data structure for transparent and repeatable reporting of bibliographic searching. *Campbell Systematic Reviews*, *18*(4), 1–12. https://doi.org/10.1002/CL2.1288

Hiebl, M. R. W. (2023). Sample selection in systematic literature reviews of management research. *Organizational Research Methods*, *26*(2), 229–261. https://doi.org/10.1177/1094428120986851

Li, Z., & Rainer, A. (2023). Reproducible searches in systematic reviews: An evaluation and guidelines. *IEEE Access*, *11*, 84048–84060. https://doi.org/10.1109/ACCESS.2023.3299211

O'Connor, A. M., Tsafnat, G., Gilbert, S. B., Thayer, K. A., & Wolfe, M. S. (2018). Moving toward the automation of the systematic review process: A summary of discussions at the second meeting of international collaboration for the automation of systematic reviews (ICASR). *Systematic Reviews*, *7*(1). https://doi.org/10.1186/S13643-017-0667-4

Scells, H., Zuccon, G., Koopman, B., & Clark, J. (2020). Automatic boolean query formulation for systematic review literature search. *WWW '20: Proceedings of the Web Conference 2020*. https://doi.org/10.1145/3366423.3380185

Schnickmann, K. (2025). *Validating and parsing academic search queries: A design science approach* [Thesis]. Otto-Friedrich-University of Bamberg.

Singh, P., & Singh, K. (2017). Exploring automatic search in digital libraries -a caution guide for systematic reviewers. *International Conference on Evaluation and Assessment in Software Engineering*. https://doi.org/10.1145/3084226.3084275

Sturm, B., & Sunyaev, A. (2019). Design principles for systematic search systems: A holistic synthesis of a rigorous multi-cycle design science research journey. *Business & Information Systems Engineering*, *61*(1), 91–111. https://doi.org/10.1007/S12599-018-0569-6

Wagner, G., & Prester, J. (2025). *CoLRev: An open-source environment for collaborative reviews* (Version 0.14.0). https://doi.org/10.5281/zenodo.11668338

Wanner, A., & Baumann, N. (2019). Design and implementation of a tool for conversion of search strategies between PubMed and ovid MEDLINE. *Research Synthesis Methods*, *10*(2), 154–160. https://doi.org/10.1002/jrsm.1314

White, J. (2024). searchRxiv: A resource for sharing database search strategies. *Medical Reference Services Quarterly*, *43*(1), 72–79. https://doi.org/10.1080/02763869.2024.2286856