Package: openalexSnapshot (via r-universe)

November 20, 2026

Title OpenAlex Bulk Snapshot Conversion, Indexing, and Record Extraction

Version 0.3.6

Description Provides tools for working with the OpenAlex bulk snapshot: converting .json.gz files to Parquet format, building ID, DOI and inverted-citation indexes over the resulting corpus, and extracting records by OpenAlex ID or DOI. Also provides an offline view of the citation graph, so the works citing or cited by a given paper can be retrieved without network access. Indexing and extraction are implemented in R over DuckDB. The package is pure R: no compiled code, and no Rust toolchain is needed to install it.

License MIT + file LICENSE

URL https://github.com/openalexPro/openalexSnapshot, https://openalexpro.github.io/openalexSnapshot/, https://doi.org/10.5281/zenodo.20448992

BugReports https://github.com/openalexPro/openalexSnapshot/issues

Additional_repositories https://openalexpro.r-universe.dev

Encoding UTF-8

Depends R (>= 4.1.0)

Imports arrow, DBI, dplyr, duckdb, future, future.apply, progressr, rlang

Suggests knitr, pak, quarto, rmarkdown, testthat (>= 3.0.0), withr

VignetteBuilder quarto

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.0.0

Config/pak/sysreqs cmake libxml2-dev libssl-dev

Repository https://openalexpro.r-universe.dev

Date/Publication 2026-11-20 08:21:52 UTC

RemoteUrl https://github.com/openalexPro/openalexSnapshot

RemoteRef main

RemoteSha 1d01985231be0032f3f81360252667d2cecf7645

Contents

build_citation_index 2
build_corpus_index 4
build_doi_index 7
doi_to_id 8
get_citing 9
lookup_by_doi 12
lookup_by_id 13

Index **16**

`build_citation_index` *Build an inverted citation index*

Description

Inverts the `referenced_works` column of the works corpus into an edge set keyed by the **cited** work, so that "which works cite X?" becomes a point lookup instead of a full-corpus scan. This is what makes `get_citing()` possible offline: the snapshot's `cited_by_api_url` is a URL and useless without network access, but the citing direction is recoverable by inverting the references every work already carries.

Usage

```
build_citation_index(
  root_dir = NULL,
  data_sets = "works",
  workers = NULL,
  memory_limit = NULL,
  retry_memory_limit = "12GB",
  temp_dir = NULL,
  block_size = 1e+07,
  batch_bytes = 1e+09,
  compression = "zstd",
  overwrite = FALSE,
  verbose = TRUE,
  corpus_dir = NULL
)
```

Arguments

root_dir	Root directory containing a parquet/ subdirectory, or the parquet directory itself.
data_sets	Datasets to index. Only works carry referenced_works, so the default is "works".
workers	Number of parallel workers.
memory_limit	DuckDB memory limit per worker, e.g. "8GB".
retry_memory_limit	Memory limit for the sequential retry of batches that exceeded memory_limit during the parallel pass. Peak memory depends on how reference-dense a batch's works happen to be, which varies enough across the corpus that no single per-worker limit suits every batch; rather than sizing memory_limit for the worst batch and throttling all of them, dense batches are re-run alone with the whole machine.
temp_dir	Directory for Stage-1 shards and DuckDB spill. Defaults to a subdirectory of tempdir(), which is on local disk, named after the index and a hash of the corpus path, so that only a re-run of the same build resumes from it. The default matters. Spilling beside the index puts those writes on the same device the corpus is being read from, and on an external USB SSD that measured **8.18 s/batch versus 0.76 s/batch** – a 10.8x difference from this setting alone. Override it only to point at a *different* fast disk, or if the default lacks room: peak usage is roughly the size of the finished index plus its transient shards.
block_size	Width of a cited_block. The default 1e7 yields ~351 partitions over the full corpus, averaging tens of MB each. Do **not** use the ID index's floor(n / 1e4): the largest OpenAlex work ID would give over 700,000 partitions.
batch_bytes	Approximate bytes of source parquet per Stage-1 batch. Batching is by byte budget over the flat file list rather than by hive partition, because the corpus is extremely skewed – one updated_date= partition can hold half the works while hundreds hold megabytes. The default of 1 GB is measured, not guessed. Unnesting a batch and writing it across hundreds of partitions costs far more memory than the few MB it emits, so the batch size governs spill. On the dense files of the real corpus, at memory_limit = "3GB": a 1 GB batch takes ~2.4 s with **no spill**; a 2 GB batch takes ~15 s; a 4 GB batch runs **out of memory**. An earlier 8 GB default spilled 23-31 GB per batch and was roughly six times slower overall. Raise this only alongside memory_limit, and measure.
compression	Parquet compression codec.
overwrite	Rebuild an existing index.
verbose	Print progress.
corpus_dir	Explicit path to a single dataset Parquet directory.

Details

get_cited() does **not** need this index – the works a paper references are simply its own referenced_works – so only one index is built, not a pair.

Each part file holds:

cited_id The cited work's OpenAlex ID, long form VARCHAR

citing_id The citing work's OpenAlex ID, long form VARCHAR

sorted by (cited_id, citing_id), matching the long-form convention of <dataset>_id_idx/ and <dataset>_doi_idx.parquet so every index in the family stores IDs the same way. Note this costs real space at three billion rows: citing_id is effectively random within a block and so does not dictionary-compress, unlike the sorted cited_id.

Value

Invisibly, the path to the index directory (corpus_dir mode) or root_dir.

Layout

Like <dataset>_id_idx/, this index is a **hive-partitioned directory** (only <dataset>_doi_idx.parquet remains a single file):

```
works_cite_idx/
  _index_meta.parquet
  cited_block=0/part-0.parquet
  cited_block=1/part-0.parquet
  ...
```

The reason is scale. The full OpenAlex works corpus holds roughly three billion reference edges; a single sorted file would need a global sort of that many rows, and its footer alone would cost tens of megabytes to parse on every query. Partitioning by a coarse block of the cited ID keeps each part small enough to sort in memory and lets a lookup open only the handful of parts it needs. _index_meta.parquet is written **last**; its presence is what marks the index complete, and it records the block_size the query side must use.

See Also

get_citing(), get_cited()

---

build_corpus_index *Build a Parquet ID-lookup index*

Description

Builds a <dataset>_id_idx/ index from the Parquet corpus produced by the snapshot download and conversion pipeline, enabling fast record retrieval by OpenAlex ID using lookup_by_id().

Usage

```
build_corpus_index(
  root_dir = NULL,
  data_sets = NULL,
  workers = NULL,
  memory_limit = NULL,
  temp_dir = NULL,
  batch_bytes = 1e+09,
  block_size = NULL,
  overwrite = FALSE,
  verbose = TRUE,
  corpus_dir = NULL,
  backend = c("auto", "r", "rust")
)
```

Arguments

root_dir	Root directory containing a parquet/ subdirectory produced by the snapshot conversion pipeline. If provided, the index for each dataset in data_sets is created at <root_dir>/parquet/<dataset>_id_idx/.
data_sets	Character vector of dataset names to index (e.g. c("works", "authors")). NULL indexes all datasets found under <root_dir>/parquet/, skipping hidden folders and index folders (names ending in _idx). Ignored when corpus_dir is provided.
workers	Number of parallel workers for Stage 1 indexing. Default is NULL (sequential).
memory_limit	DuckDB memory limit (e.g., "20GB"). Default is NULL.
temp_dir	Directory for Stage-1 shards and DuckDB spill. Defaults to a subdirectory of tempdir(), which is on local disk, named after the index and a hash of the corpus path, so that only a re-run of the same build resumes from it. The default matters. Spilling beside the index puts those writes on the same device the corpus is being read from, and on an external USB SSD that measured **8.18 s/batch versus 0.76 s/batch** – a 10.8x difference from this setting alone. Override it only to point at a *different* fast disk, or if the default lacks room: peak usage is roughly the size of the finished index plus its transient shards.
batch_bytes	Approximate bytes of source parquet per Stage-1 batch.
block_size	Width of an id_block. NULL (default) derives one from the actual id range so the index lands near 300 blocks. A fixed width does not suit every dataset, because entity id ranges differ by orders of magnitude. At 1e7, works gives 351 useful blocks but authors gives only **14** – their ids cluster tightly around 5.0-5.1e9, so 113 million rows would land in 14 partitions of ~8 M rows and prune almost nothing. Pass a value explicitly to override.
overwrite	If TRUE, rebuilds existing indexes. Default is FALSE (skip if the index already exists).
verbose	Print progress messages. Default is TRUE.

`corpus_dir` Explicit path to a single dataset Parquet directory (e.g. `"/Volumes/openalex/parquet/works"`). The index is written as a sibling directory: `<parent>/<basename>_id_idx/`. When this is provided, `root_dir` and `data_sets` are ignored.

`backend` Retained only so that existing calls passing `backend = "rust"` get an explanatory error. The compiled backend was removed in 0.1.0; the package is pure R. `"auto"` (the default) and `"r"` both use the pure-R/DuckDB implementation. `"rust"` uses the compiled library and is **deprecated**: it writes an unsorted index and supports neither `columns` nor `add_columns`. It will be removed in a future release. `"auto"` (the default) uses the compiled Rust library when it is loaded and the pure-R/DuckDB implementation otherwise, so behaviour is unchanged for an installed binary. `"r"` forces pure R and is always available. `"rust"` forces the compiled path and errors if it is not loaded.

Details

The function uses a two-stage approach:

1. Index each Parquet file individually (bounded memory, parallel, with resume support).
2. Combine the per-file shard indexes into a single Parquet index.

Paths can be supplied as a single `root_dir` (which iterates over all requested `data_sets`) or as an explicit `corpus_dir` pointing to a single dataset directory.

A hive-partitioned directory, not a single file:

```
works_id_idx/
  _index_meta.parquet
  id_block=0/part-0.parquet
  ...
```

with `id_block = floor(numeric_id / block_size)`, each part sorted by `id`, holding `id` (long form), `parquet_file` (relative to the parquet root) and `file_row_number` (0-indexed).

A single file measured 3,992 row groups whose footer cost 0.192 s to parse before reading any data, on every query, and needed a global sort of 492M rows to build. Partitioning removes both: blocks sort independently, and a lookup opens only the blocks its ids fall in.
`_index_meta.parquet` is written last; its presence marks the index complete, and it records the `block_size` the query side must recompute.

Value

When `corpus_dir` is provided, invisibly returns the path to the created index file. When `root_dir` is used, invisibly returns `root_dir`.

See Also

`lookup_by_id()` for ID-based record retrieval, `build_citation_index()` and `build_doi_index()`.

`build_doi_index` *Build a Parquet DOI-lookup index*

Description

Builds a `<dataset>_doi_idx.parquet` index mapping normalised DOIs to OpenAlex IDs and to their location in the Parquet corpus, so DOIs can be resolved without network access.

Usage

```
build_doi_index(
  root_dir = NULL,
  data_sets = "works",
  workers = NULL,
  memory_limit = NULL,
  temp_dir = NULL,
  batch_bytes = 8e+09,
  compression = "zstd",
  overwrite = FALSE,
  verbose = TRUE,
  corpus_dir = NULL
)
```

Arguments

`root_dir`	Root directory containing a `parquet/` subdirectory, or the parquet directory itself.
`data_sets`	Datasets to index. Only `works` carries a `doi` column, so the default is `"works"`.
`workers`	Number of parallel workers for Stage 1.
`memory_limit`	DuckDB memory limit, e.g. `"20GB"`.
`temp_dir`	Directory for Stage-1 shards and DuckDB spill. Defaults to a subdirectory of `tempdir()`, which is on local disk, named after the index and a hash of the corpus path, so that only a re-run of the same build resumes from it. The default matters. Spilling beside the index puts those writes on the same device the corpus is being read from, and on an external USB SSD that measured **8.18 s/batch versus 0.76 s/batch** – a 10.8x difference from this setting alone. Override it only to point at a *different* fast disk, or if the default lacks room: peak usage is roughly the size of the finished index plus its transient shards.
`batch_bytes`	Approximate bytes of source parquet per Stage-1 batch.
`compression`	Parquet compression codec.
`overwrite`	Rebuild an existing index.
`verbose`	Print progress.
`corpus_dir`	Explicit path to a single dataset Parquet directory. When given, `root_dir` and `data_sets` are ignored.

Details

The corpus stores DOIs as full lower-cased resolver URLs (`https://doi.org/10.1097/...`). The index key is the **bare** DOI, so that a caller may supply any of these and get the same answer:

```
10.1016/j.joi.2017.08.007
https://doi.org/10.1016/j.joi.2017.08.007
http://dx.doi.org/10.1016/j.joi.2017.08.007
doi:10.1016/j.joi.2017.08.007
DOI: 10.1016/J.JOI.2017.08.007
```

Carrying `parquet_file` and `file_row_number` alongside the ID means `lookup_by_doi()` can extract full records from one index read rather than two.

The index contains:

doi Normalised bare lower-case DOI, no resolver prefix

id OpenAlex ID in long form

id_block `floor(numeric_id / 10000)`, as in the ID index

parquet_file Path relative to the parquet root

file_row_number Row number within that file (0-indexed)

Rows are sorted by `doi`, so a point lookup reads the footer plus one row group rather than scanning the file.

Value

Invisibly, the path to the index file (`corpus_dir` mode) or `root_dir`.

See Also

`doi_to_id()`, `lookup_by_doi()`, `build_corpus_index()`

`doi_to_id` *Resolve DOIs to OpenAlex IDs offline*

Description

Looks DOIs up in the index built by `build_doi_index()`. DOIs may be given with or without a resolver prefix; all of these resolve identically:

Usage

```
doi_to_id(
  doi,
  root_dir = NULL,
  index_file = NULL,
  data_set = "works",
  verbose = TRUE
)
```

Arguments

doi	Character vector of DOIs.
root_dir	Root directory containing a parquet/ subdirectory, or the parquet directory itself.
index_file	Explicit path to a *_doi_idx.parquet. Overrides root_dir.
data_set	Dataset name used to locate the index under root_dir.
verbose	Print progress.

Details

```
10.1016/j.joi.2017.08.007
https://doi.org/10.1016/j.joi.2017.08.007
http://dx.doi.org/10.1016/j.joi.2017.08.007
doi:10.1016/j.joi.2017.08.007
DOI: 10.1016/J.JOI.2017.08.007
```

Value

A data frame with one row per input, in input order:

doi_input The string as supplied

doi Its normalised key

id The OpenAlex ID in long form, or NA if unresolved

A data frame rather than a bare vector, because the caller needs to know *which* inputs failed to resolve.

See Also

build_doi_index(), lookup_by_doi()

---

get_citing *Find the works citing, or cited by, given keypapers*

Description

get_citing() returns the works that **cite** the keypapers; get_cited() returns the works the keypapers **reference**. Both work entirely offline against a local snapshot.

Usage

```
get_citing(
  keypaper,
  root_dir = NULL,
  return = c("edges", "ids", "records"),
  output = NULL,
  columns = NULL,
  max_results = 100000L,
  depth = 1L,
  citation_index = NULL,
  id_index = NULL,
  doi_index = NULL,
  data_set = "works",
  workers = NULL,
  memory_limit = NULL,
  temp_dir = NULL,
  verbose = TRUE
)

get_cited(
  keypaper,
  root_dir = NULL,
  return = c("edges", "ids", "records"),
  output = NULL,
  columns = NULL,
  max_results = 100000L,
  depth = 1L,
  id_index = NULL,
  doi_index = NULL,
  data_set = "works",
  workers = NULL,
  memory_limit = NULL,
  temp_dir = NULL,
  verbose = TRUE
)
```

Arguments

keypaper	Character **vector** of one or more keypapers. Forms may be mixed freely in a single call: • short OpenAlex ID, `"W3045921891"` • long OpenAlex ID, `"https://openalex.org/W3045921891"` • bare DOI, `"10.1016/j.joi.2017.08.007"` • DOI with resolver, `"https://doi.org/10.7717/peerj.4375"` All keypapers are resolved and queried in one pass, not in a loop.
root_dir	Root directory containing a `parquet/` subdirectory, or the parquet directory itself.

return	What to return: • "edges" (default) – a data frame of from/to long-form IDs, where a row A, B means **A cites B**. This keeps the mapping between each keypaper and its neighbours, which a flat ID vector would lose when keypaper has length > 1, and it is directly rbind-able into a snowball edge set. • "ids" – the unique non-keypaper endpoint, as a character vector. • "records" – the full records for those IDs, via lookup_by_id().
output	Directory to write records to when return = "records". When NULL, records are returned as a data frame.
columns	Column projection passed to lookup_by_id() when return = "records".
max_results	Error if more edges than this are found. A single heavily cited work can have hundreds of thousands of citing works, and the follow-on record extraction would then touch essentially every file in the corpus. Set Inf to lift the guard deliberately.
depth	Reserved for multi-hop snowballing; only 1 is implemented.
citation_index	Explicit path to a *_cite_idx directory.
id_index	Explicit path to a *_id_idx directory.
doi_index	Explicit path to a *_doi_idx.parquet.
data_set	Dataset name used to locate indexes under root_dir.
workers	Parallel workers for record extraction.
memory_limit	DuckDB memory limit.
temp_dir	Root for DuckDB's spill files. NULL (default) uses a private directory under tempdir(). Never DuckDB's own default, which is .tmp relative to the working directory and is therefore shared by every concurrent caller.
verbose	Print progress.

Details

The two are not symmetric in cost. get_cited() only needs the keypaper's own referenced_works column, so it uses the ID index alone. get_citing() asks the opposite question, which the snapshot does not answer directly – cited_by_api_url is a URL – so it needs the inverted index from build_citation_index().

Value

A data frame of edges, a character vector of IDs, or records – see return.

See Also

build_citation_index(), build_doi_index(), lookup_by_id()

lookup_by_doi *Look up records by DOI*

Description

Convenience wrapper: doi_to_id() followed by lookup_by_id().

Usage

```
lookup_by_doi(
  doi,
  root_dir = NULL,
  index_file = NULL,
  id_index = NULL,
  data_set = "works",
  verbose = TRUE,
  ...
)
```

Arguments

doi	Character vector of DOIs.
root_dir	Root directory containing a parquet/ subdirectory, or the parquet directory itself.
index_file	Explicit path to a *_doi_idx.parquet. Overrides root_dir.
id_index	Explicit path to a *_id_idx directory.
data_set	Dataset name used to locate the index under root_dir.
verbose	Print progress.
...	Passed to lookup_by_id(), e.g. columns or output.

Value

Whatever lookup_by_id() returns.

See Also

doi_to_id(), lookup_by_id()

lookup_by_id *Look up records by OpenAlex ID*

Description

Uses a pre-built index (created by `build_corpus_index()`) to locate records efficiently and extract them from the Parquet corpus.

Usage

```
lookup_by_id(
  root_dir = NULL,
  ids,
  project_dir = NULL,
  data_sets = NULL,
  workers = NULL,
  progress = TRUE,
  verbose = TRUE,
  index_file = NULL,
  output = NULL,
  backend = c("auto", "r", "rust"),
  columns = NULL,
  add_columns = NULL,
  memory_limit = NULL,
  temp_dir = NULL
)
```

Arguments

`root_dir`	Root directory containing `parquet/` and the dataset indexes produced by `build_corpus_index()`. Index files are expected at `<root_dir>/parquet/<dataset>_id_idx/`.
`ids`	Character vector of OpenAlex IDs to retrieve. Can be long form (e.g. `"https://openalex.org/W274180` or short form (e.g. `"W2741809807"`). Duplicates are dropped: each id is returned at most once. If the index holds an id in more than one source file, the record from the latest `updated_date=` partition is kept.
`project_dir`	Project output directory. Extracted Parquet files are written to `<project_dir>/snapshot_extract_<dat` Only used when `root_dir` is provided.
`data_sets`	Character vector of dataset names to search (e.g. `c("works", "authors")`). `NULL` searches all indexed datasets under `<root_dir>/parquet/`. Ignored when `index_file` is provided.
`workers`	Number of parallel workers for reading corpus files. Default is `NULL` (sequential).
`progress`	Ignored (kept for backward compatibility).
`verbose`	Print progress messages. Default is `TRUE`.

index_file — Explicit path to an index Parquet file created by build_corpus_index(). When provided, root_dir, data_sets, and project_dir are ignored.

output — Path to an output directory for writing results as Parquet files when using index_file mode. If NULL (default), results are returned as a data frame. Ignored when root_dir is used (use project_dir instead). Each source file is written to its own part, and a file that fails to read or write is an error, not a warning.

backend — Retained only so that existing calls passing backend = "rust" get an explanatory error. The compiled backend was removed in 0.1.0; the package is pure R. "auto" (the default) and "r" both use the pure-R/DuckDB implementation. "rust" uses the compiled library and is **deprecated**: it writes an unsorted index and supports neither columns nor add_columns. It will be removed in a future release. "auto" (the default) uses the compiled Rust library when it is loaded and the pure-R/DuckDB implementation otherwise. "r" forces pure R and is always available. "rust" forces the compiled path and errors if it is not loaded.

columns — Character vector of columns to return. NULL (default) returns all. Projection matters: reading 2 of 51 columns from a corpus of nested structs is far cheaper than SELECT *. Requires backend = "r" - the compiled path always reads every column.

add_columns — Named list of constant columns to add to every returned row, e.g. list(relation = "citing"). Values are embedded as **SQL string literals**, matching openalexPro::pro_request_parq so a logical must be passed as "TRUE" and cast by the caller. Requires backend = "r".

memory_limit — DuckDB memory_limit for each worker. NULL (default) leaves DuckDB's own, which is ~80% of system RAM *per worker*.

temp_dir — Root for DuckDB's spill files. NULL (default) uses a private per-worker directory under tempdir(). Workers must not share one: concurrent DuckDB instances write colliding duckdb_temp_storage_*.tmp files and corrupt each other's spill.

Details

Paths can be supplied as a root_dir + data_sets pair (which automatically locates the correct index files and writes output into project_dir) or as an explicit index_file for direct use.

Value

- index_file mode, output not NULL: invisibly returns output.
- index_file mode, output is NULL: returns a data frame of matching records.
- root_dir mode: invisibly returns project_dir.

See Also

build_corpus_index() for building the required index.

Examples

```
## Not run:
# root_dir mode (searches multiple datasets)
lookup_by_id(
  root_dir    = "/Volumes/openalex",
  ids         = c("W2741809807", "W1234567890"),
  project_dir = "my_project",
  data_sets   = "works"
)

# index_file mode (direct access, returns data frame)
records <- lookup_by_id(
  index_file = "works_id_index.parquet",
  ids        = c("W2741809807", "W1234567890")
)

# index_file mode (write to parquet)
lookup_by_id(
  index_file = "works_id_index.parquet",
  ids        = large_id_vector,
  output     = "filtered_works",
  workers    = 3
)

## End(Not run)
```

Index

build_citation_index, 2
build_citation_index(), *6*, *11*
build_corpus_index, 4
build_corpus_index(), *8*, *13, 14*
build_doi_index, 7
build_doi_index(), *6*, *8, 9*, *11*

doi_to_id, 8
doi_to_id(), *8*, *12*

get_cited *(*get_citing*)*, 9
get_cited(), *3, 4*
get_citing, 9
get_citing(), *2*, *4*

lookup_by_doi, 12
lookup_by_doi(), *8, 9*
lookup_by_id, 13
lookup_by_id(), *4*, *6*, *11, 12*

tempdir(), *3*, *5*, *7*, *11*, *14*