OpenCódice Technical Report OC-TR-2026-011

zenodo-mcp: A Model Context Protocol Server for the Zenodo Open-Research Repository

Francisco-Javier Rodrigo-Ginés
OpenCódice Research
fran@opencodice.org

Jorge Chamorro-Padial
OpenCódice Research
jorge@opencodice.org

4 May 2026 • **DOI:** 10.5281/zenodo.20023539

Abstract

The Model Context Protocol (MCP) ecosystem now reaches arXiv, Semantic Scholar, Crossref, ORCID, OpenReview, and DBLP, but stops at scholarly metadata. The artifacts those records describe (datasets, software releases, slide decks, supplementary materials) live on a separate substrate the protocol does not yet touch. The most general home for those artifacts is Zenodo, CERN's open-research repository: every record carries a permanent DOI, every concept-DOI lets the consumer walk version history, and most artifacts ship under structured-string licenses parseable without natural-language inference. We introduce `zenodo-mcp`, an open-source MCP server exposing eight tools backed by the Zenodo REST API. We document the architecture, the deposit-versus-record API distinction, the version-walking design via concept-recid, and the deliberate non-design of bulk-binary streaming through the JSON-bound MCP transport. We then present a 100-record audit of real Zenodo deposits returned by the query "machine learning": eleven distinct license strings, eight distinct resource-types, and an aggregate permissive-licence rate of 77%. The licence heterogeneity has direct consequences for any agent that wants to filter to permissive licenses only; we discuss what an MCP wrapper has to surface to make that filtering tractable, and what future work should add.

Keywords: Model Context Protocol, Zenodo, open science, research data, datasets, FAIR data, software releases, scholarly infrastructure, license heterogeneity

Highlights

- Open-source MCP server exposing eight tools backed by the Zenodo REST API for CERN's general-purpose open-research repository
- Every record carries a permanent DOI; the server walks version history via concept-recid and surfaces structured file lists with checksums
- Read-only by design: the server reads records, files, and communities but does *not* deposit; v0.2 will add deposit-side tools through a separate, scope-limited credential
- 100-record audit of live Zenodo deposits: eleven distinct license strings, eight distinct resource-types, 67% CC-BY-4.0, 77% permissively-licensed; non-trivial filtering work for any MCP-mediated agent
- Closes a five-server scholarly-MCP loop: peer review + integrity + authors + venues + artifacts. For the first time, an MCP-connected LLM can reason about a paper as a unit of research

1 Introduction

The Model Context Protocol (MCP) [Anthropic, 2024] has matured rapidly into the default integration substrate between large language models and external systems. By the time of writing, the public registry of MCP servers covers most of the everyday research stack: scholarly search through Semantic Scholar and arXiv, dataset discovery through Hugging Face, citation resolution through Crossref, and code search through GitHub.

The largest remaining gap concerns the *artifacts* that scholarly records describe. Every academic-MCP server we are aware of treats papers as the unit of information: title, abstract, citations, PDF link. This framing is sufficient for discovery but ignores the data, code, slide decks, and supplementary materials a researcher actually needs to verify a result, re-run an experiment, or cite a specific revision of a dataset. The publisher's site is the wrong place to look for those artifacts. URLs rot, supplementary-materials links break behind paywalls, and the licensing terms attached to "Supplementary File 3" are typically unclear or non-existent [Vasilevsky et al., 2017]. For an LLM agent attempting to actually re-run the experiment described in a paper, this is a hard problem: the natural data source (the publisher) is hostile, the licence is ambiguous, and the artifact may have moved between the paper's submission and the agent's read.

Zenodo is the most general substitute [CERN / OpenAIRE, 2013–present]. Hosted by CERN since 2013 and operated as part of the European Open Science Cloud, every Zenodo record carries a permanent DOI minted by DataCite [Neumann and Brase, 2014], every concept-DOI lets the consumer walk version history, and most artifacts are deposited under open licenses with structured license strings in the metadata. This combination is exactly the property set the FAIR Guiding Principles [Wilkinson et al., 2016] call for: findability via persistent identifier, accessibility via open API, interoperability via structured metadata, and reusability via explicit license. Zenodo now hosts more than three million records spanning datasets, software releases, presentations, posters, and grey literature; the licensing field is structured rather than free-text; and the version-walking concept (one stable concept-DOI plus a sequence of version-specific DOIs) gives the consumer a clean way to pin to a specific revision while warning the user when newer versions exist.

There is no MCP server for Zenodo. This report introduces `zenodo-mcp`, an open-source MCP server that closes the gap. We make three contributions.

- **A complete read-side MCP surface for the Zenodo REST API.** Eight tools cover record search, single-record retrieval (by id and by DOI), version-history walking, file enumeration with download URLs and checksums, community search and retrieval, and creator-name search. The schema flattens nested metadata into top-level fields so the consuming LLM can filter without traversing nested objects.
- **A deliberate read-only scope.** The v0.1 server reads but does not deposit. Deposit-side tools are on the v0.2 runway under a separate, scope-limited credential, because the blast radius of an agent that can publish records to Zenodo is meaningfully larger than the blast radius of one that can only read.
- **A 100-record audit of real Zenodo deposits.** We point the server at its own backend and survey the license and resource-type distribution of the first 100 records returned by a free-text query for "machine learning". The result, summarised in Section 7, is heterogeneous in ways that have direct consequences for any agent attempting to filter to permissive licences only.

The server is released under the MIT license at `https://github.com/OpenCodice-Research/zenodo-mcp`. Its publication closes a five-server scholarly-MCP loop covering peer review, research integrity, canonical author identity, venue context, and artifacts.

The remainder of this report is organised as follows. Section 2 reviews Zenodo's role in the scholarly infrastructure stack and the concept-DOI model. Section 3 documents the system architecture. Section 4 catalogues the eight tools and the schema design. Section 5 discusses the concept-recid mechanism and the version-walking implementation. Section 6 explains the deliberate non-design of binary streaming. Section 7 presents a 100-record audit of license and resource-type distributions and the implications for agent-side filtering. Section 8 places the server within a wider family of scholarly MCP wrappers. Sections 9 and 10 discuss limitations and future directions.

2 Background

2.1 Zenodo as scholarly infrastructure

Zenodo was launched in 2013 by CERN as part of the OpenAIRE infrastructure project [CERN / OpenAIRE, 2013–present]. The design choice that distinguishes it from publisher-hosted supplementary materials, institutional repositories, and pre-print servers is *permanent identity at the per-deposit level*. Every record receives a unique DOI minted through DataCite [Neumann and Brase, 2014]; the DOI resolves to the Zenodo record page; the record page lists files with download URLs and structured metadata. The pattern is the same as Crossref's DOI minting for journal articles, but applied to anything a researcher wants to deposit: datasets, code releases, slide decks, supplementary materials, even preprints.

Adoption has been steady. The repository now hosts more than three million records and is integrated by GitHub (every tagged release of an opted-in GitHub repository is automatically deposited), by major funders' open-data mandates (the European Research Council, NIH, NASA), and by an increasing number of scholarly publishers as the canonical home for supplementary materials. The trajectory follows the broader push toward FAIR-compliant open data [Stall et al., 2019] and aligns with the recommendations of the data-citation roadmap for scientific publishers [Cousijn et al., 2018].

2.2 The concept-DOI model

A central design feature is the distinction between *concept-DOI* and *version-DOI*. When a researcher first deposits a record, Zenodo mints two DOIs simultaneously: a stable concept-DOI representing "this artifact regardless of version" and a version-DOI representing "version 1 of this artifact". Subsequent revisions mint new version-DOIs but reuse the concept-DOI. The convention is that the concept-DOI always resolves to the latest version, while a version-DOI always resolves to a specific revision.

For a citing author, the choice depends on intent. If the citation is to "the dataset" as an evolving artifact, the concept-DOI is appropriate. If the citation is to "version 1.2.3 of the dataset, the one used in the experiment described in this paper", the version-DOI is appropriate. `zenodo-mcp` surfaces both: `Record.doi` carries the version-DOI, `Record.concept_doi` carries the concept-DOI, and `zenodo_get_versions` walks the full version sequence. The two-layer scheme is precisely the kind of structured citation primitive software-citation principles call for [Smith et al., 2016].

2.3 Existing academic MCP servers

The MCP catalogue covers paper-level metadata (arXiv, Semantic Scholar, Crossref, ACL Anthology, OpenReview, Retraction Watch, ORCID, DBLP), but not artifacts. Hugging Face's hub and Kaggle have dedicated MCP wrappers, but neither covers the long tail of dataset deposits that live on Zenodo, and neither exposes the concept-DOI model. The omission is structural: paper-discovery wrappers were built first because the integration was easier; artifact wrappers are harder because heterogeneous metadata, heterogeneous licences, and heterogeneous file types

each force the wrapper to do more normalisation work [Pasquetto et al., 2017].

3 System architecture

`zenodo-mcp` is a small Python package built around four concerns: a thin client wrapper over the Zenodo REST API, a disk-backed cache, a set of pure-function tool implementations, and a FastMCP server that registers them as MCP tools.

3.1 Layered design

1. **Client wrapper** (`zenodo_mcp.client`). Lazy-instantiates an `httpx.Client` against `zenodo.org/api`. With `ZENODO_TOKEN` set, the wrapper sends the token as a Bearer header, granting higher rate limits and access to records the token's owner has read access to. Without it, the wrapper runs against the public read-only tier (with the lower per-page cap; see Section 7).

2. **Cache** (`zenodo_mcp.cache`). Diskcache-backed key-value store with six-to-twenty-four-hour TTLs.

3. **Schemas** (`zenodo_mcp.schemas`). Pydantic v2 models: `Creator`, `FileEntry`, `Record`, `CommunitySummary`.

4. **Tools** (`zenodo_mcp.tools`). Pure functions grouped by purpose: `records.py` (search, get, versions, files, by-DOI, by-creator), `communities.py` (community search and retrieval).

5. **Server** (`zenodo_mcp.server`). FastMCP application registering each tool with an `@mcp.tool()` decorator.

6. **Transport** (`zenodo_mcp.cli`). Argparse-based CLI selects between `stdio` and `streamable-http`.

3.2 The deposit-versus-record API distinction

Zenodo exposes two parallel APIs that share the same database. The `/api/records` endpoint is the read-side view: it returns published records with their files, metadata, and links, and is the only API path `zenodo-mcp` uses. The `/api/deposit/depositions` endpoint is the write-side view: it lets an authenticated client create draft records, upload files, set metadata, and publish.

The two APIs return slightly different shapes for the same underlying record. The deposit endpoint returns a `links.bucket` field for direct file uploads; the record endpoint surfaces files inline. v0.1 of `zenodo-mcp` reads through `/api/records` exclusively; the v0.2 deposit-side tools will use `/api/deposit/depositions` under a separate, scope-limited credential.

3.3 File metadata and the structured license string

Zenodo metadata is comparatively rich. Every record carries a structured `license` field (one of CC0, CC-BY, CC-BY-SA, CC-BY-NC, MIT, Apache-2.0, GPL variants, or proprietary), a `resource_type` field (`dataset`, `software`, `publication`, `presentation`, `poster`), and a structured `publication_date`. The `Record` schema flattens these into top-level fields so the consumer can filter without traversing nested objects.

Files carry a `key` (filename), `size` (bytes), `checksum` (typically MD5), and `links.self` (a content URL that streams the file when fetched). `zenodo-mcp` surfaces all four on every `FileEntry`, so an agent can validate the integrity of a downloaded file without a second API call.

4 Tools

The server exposes eight tools grouped into two families. Table 1 summarises the surface.

Family	Tool	Purpose
Records	`zenodo_search`	Free-text search with type, community, sort filters
	`zenodo_get_record`	Full record by numeric record id
	`zenodo_get_record_by_doi`	Resolve a DOI to its Zenodo record
	`zenodo_get_versions`	Walk every version of a record via concept-recid
	`zenodo_get_files`	List files with download URLs and checksums
	`zenodo_search_by_creator`	Records by creator name
Communities	`zenodo_search_communities`	Community search by name or description
	`zenodo_get_community`	Community by slug or id

Table 1: The eight tools exposed by `zenodo-mcp`, grouped by family. All tools are prefixed with `zenodo_` when registered as MCP tools.

4.1 Schema design

Three choices are worth highlighting.

License as a string, not an object. Zenodo's record metadata wraps the licence as either a string (the licence id, e.g. `cc-by-4.0`) or a small object with id and URL. The `Record` schema flattens both shapes to a single string, picking `license.id` when the object form is used. The lower-cased string is what the agent filters on; downstream code rarely needs the URL.

Communities as a list of slugs. A record may belong to multiple Zenodo communities (a research-area community, an institutional community, a funder-mandated community). The schema flattens these to a list of slug strings; downstream filtering by community is a simple string match.

Files surfaced inline on `get_record`. `zenodo_get_record` returns the full record *including* its files. `zenodo_search`, by contrast, returns records without files (for response size). Consumers that need the files invoke `zenodo_get_record` or `zenodo_get_files` after the search.

5 Version walking via concept-recid

The version-walking story is the most interesting design choice. When a Zenodo record exists in multiple versions, the `conceptrecid` field on any version points to the shared concept record (a phantom record that represents "the artifact across versions"). To walk all versions, the consumer issues a search for `conceptrecid:<id>` with the `all_versions=true` flag.

5.1 Implementation

```
def get_versions(client: ZenodoClient, record_id: int) -> list[Record]:
    record = client.get(f"/api/records/{record_id}")
    concept = record.get("conceptrecid")
    if not concept:
        return [parse_record(record)]
    payload = client.get(
        "/api/records",
        params={
            "q": f"conceptrecid:{concept}",
            "all_versions": "true",
```

```
            "size": 100,
            "sort": "version",
        },
    )
    hits = payload.get("hits", {}).get("hits", [])
    return [parse_record(h) for h in hits]
```

The function takes any version-id of a record (the input could be the record id of v1, v3, or the latest), resolves to the concept-recid, and walks all versions sorted by version number. The result is a list of `Record` objects, each carrying its own version-DOI, alongside the shared concept-DOI.

5.2 Why this matters for citation

A citing author who points to "the latest version" of an evolving dataset risks a citation that drifts: today's reference is to v3, next year's reader sees v7, and the version mismatch can invalidate any reproducibility claim that depended on the specific data the author actually used. `zenodo_get_versions` surfaces the full version list, so the citing author (or the agent doing the citing) can pin to a specific version-DOI explicitly. The concept-DOI remains available for citations that are intentionally version-agnostic. This is the structured citation primitive software-citation principles call for: a citation that is both stable and discoverable, with the choice between the two made explicit at the moment the citation is generated [Smith et al., 2016, Cousijn et al., 2018].

> **Key insight.** The concept-DOI / version-DOI distinction is one of Zenodo's quietest and most important design choices. It lets a citation be both stable (the version-DOI never drifts) and discoverable (the concept-DOI always resolves to something useful). `zenodo_get_versions` surfaces both so an LLM agent can pin or float as appropriate.

6 The non-design of binary streaming

A natural extension of the file-enumeration tool is to fetch the file bytes directly. `zenodo-mcp` does not do this, by design.

The MCP transport (both `stdio` and `streamable-http`) is built around JSON request/response. Tool results are JSON-serialisable and bounded in size by client limits (typically a few MB). Fetching a 12 GB dataset through a JSON tool result is the wrong shape entirely: the transport is not built for it, the consuming LLM cannot reason over raw binary bytes, and the result inflates the agent's context window catastrophically.

The right shape is to return the URL and let the consumer fetch the bytes out-of-band. `zenodo_get_files` returns the structured `FileEntry` with the `download_url` and `checksum`; the consumer (a script, a CI job, a dataset-loading library) handles the download itself.

We document this non-design here because it is a recurring source of confusion. "Why doesn't `zenodo_get_files` return the file contents?" is a question we expect to receive; the answer is that the MCP transport is the wrong layer for binary streaming, and the right answer is composition: `zenodo-mcp` surfaces metadata + URL, the consumer streams. v0.2 may add an optional inline-fetch mode for very small files (a few hundred kB), but the default will remain URL-only.

7 An audit of real Zenodo deposits

A thin wrapper deserves more than a hello-world demonstration. To stress-test the schema and to surface a finding interesting enough to motivate the wrapper's existence, we run `zenodo-mcp`

against its own backend and survey 100 real records. The exercise probes two questions: how heterogeneous is the license metadata an MCP-mediated agent has to filter on, and how heterogeneous is the resource-type metadata such an agent has to dispatch on.

7.1 Setup

We issue four paginated calls to `zenodo_search` with the free-text query "machine learning", no resource-type filter, and the anonymous-tier page size cap of 25. The pagination is forced: the public Zenodo API caps anonymous `size` at 25 and rejects larger requests with a 400 error, citing the limit and pointing the caller at authenticated requests for higher caps. This is a quiet but important wrapper-layer concern: the consuming agent should not have to discover the cap by trial and error, and a future wrapper version should auto-paginate when the requested size exceeds the anonymous limit.

For each of the resulting 100 records we read the flattened `license`, `resource_type`, `doi`, and `concept_doi` fields. We also pull the version sequence for the top hit through `zenodo_get_versions`; that call surfaced a second public-tier limit (the version-walking helper hard-codes `size=100`, which the public API rejects under the same cap). Both observations feed back into Section 9 and the v0.2 runway.

7.2 License distribution

Table 2 reports the licence distribution across the 100 records. Eleven distinct license strings appear. CC-BY-4.0 dominates with 67% of records, but the long tail is non-trivial: ten records carry the catch-all `other-open` string (a Zenodo umbrella for any license the depositor declared as open without picking from the canonical list), seven carry `mit-license`, four carry `cc-by-nc-sa-4.0` (a non-commercial, share-alike variant that an agent re-using the artifact for a commercial purpose cannot accept), and three records carry no machine-readable license at all.

License id	Records
`cc-by-4.0`	67
`other-open` (umbrella)	10
`mit-license`	7
`cc-by-nc-sa-4.0`	4
`cc-by-sa-4.0`	3
`[unknown]` (no license recorded)	3
`apache2.0`	2
`cc-by` (unversioned)	1
`bsd-3-clause`	1
`gpl-3.0-or-later`	1
`cc-zero`	1
Total	100

Table 2: License distribution across 100 Zenodo records returned by the query "machine learning" on 4 May 2026. Eleven distinct license strings appear, CC-BY-4.0 dominates, the tail is non-trivial.

The headline finding has direct consequences for any LLM agent attempting to filter to permissive-only artifacts.

Aggregate openness vs. aggregate permissiveness. Treating CC0, CC-BY-X (any version), MIT, Apache, BSD, ISC, GPL-X, LGPL-X, AGPL, MPL, EPL, and the various CC-BY-NC and CC-BY-SA variants as "open" yields 85 of the 100 records. Restricting to *permissive* (CC0,

CC-BY-X, MIT, Apache, BSD, ISC) leaves only 77. The 8-point gap is exactly the population an agent has to be careful about: copyleft and non-commercial licences are technically open but technically incompatible with many downstream agent uses. An agent that conflates "open" with "permissive" will silently violate the licence terms of roughly one in ten records it touches.

Three records carry no license at all. The three `[unknown]` records are records on which the depositor did not pick a license. Under copyright defaults, these records are *all rights reserved* unless a licence is declared elsewhere in the deposit. An MCP-mediated agent that treats missing-license as "presumably open" is wrong, and the wrapper layer should make missing-license loud rather than quiet. The `Record` schema's choice to surface the licence as `None` when absent (rather than substituting an empty string or a default) is what enables the consuming LLM to flag these records distinctly.

The `other-open` umbrella is not actionable. Ten records carry the `other-open` string. This is Zenodo's catch-all for licences that are open but not on Zenodo's curated list (custom institutional licences, unusual Creative Commons variants, dual-licensed code). For an agent that wants to make a yes/no permissive-license decision, `other-open` is not actionable: the agent would have to follow up with a fetch of the licence URL or a free-text licence string from the record description. This is the kind of metadata gap an MCP wrapper cannot fix unilaterally; the right answer is to surface `other-open` alongside a follow-up tool call `zenodo_get_record` that reads the description for any custom-licence text.

Older un-versioned strings linger. One record carries the bare `cc-by` string (no version), a depositor convention from before Zenodo enforced versioned-licence ids. The wrapper surfaces it as-is; the agent has to know that bare `cc-by` is approximately CC-BY-4.0 but not exactly, and choose how strict to be about version pinning. Future wrapper work should normalise these legacy strings.

7.3 Resource-type distribution

Table 3 reports the resource-type distribution across the same 100 records. Eight distinct types appear: `publication` (45), `software` (30), `dataset` (15), `model` (4), `presentation` (2), `poster` (2), `image` (1), `other` (1).

Resource type	Records
`publication`	45
`software`	30
`dataset`	15
`model`	4
`presentation`	2
`poster`	2
`image`	1
`other`	1
Total	100

Table 3: Resource-type distribution across the same 100 records. Eight distinct types appear, a publications-heavy long tail dominates the response, datasets and software trail.

The result is non-obvious. The query was "machine learning", which most readers would expect to bias toward datasets and software. In practice the dominant type is `publication` (45%):

preprints, conference papers, and grey-literature reports deposited on Zenodo as the canonical archival home. Datasets are only 15% of the response; software is 30%; the recently-introduced `model` type (for trained model checkpoints) is 4%. An MCP-mediated agent that wants to find "a dataset for a machine-learning experiment" has to filter post-hoc on `resource_type`; the bare query does not surface datasets preferentially.

7.4 Other observations

Every record carries a DOI. All 100 records carry a version-level DOI in `Record.doi`; 85 also carry a concept-DOI in `Record.concept_doi`. The 15-record gap is records that exist in only one version (the concept and version are conceptually distinct but practically equal; Zenodo populates only the version-DOI in this case).

Files are not surfaced on search. Zero of the 100 records carry a populated `files` list in the search response, by design. `zenodo_search` returns records without files for response-size reasons; the consumer fetches files via a follow-up `zenodo_get_record` or `zenodo_get_files` call. The schema design makes the absent-files distinction obvious (an empty list, not a missing key), but a downstream agent that does not read the docs may assume files are absent and skip the follow-up.

The version-walking call hits the same cap as search. The version-walking helper hard-codes `size=100` on the underlying search request, which the anonymous tier rejects. For records with multiple versions, an unauthenticated caller cannot walk the full version sequence today; this is a v0.1 limitation the v0.2 release will fix by either auto-detecting the cap or paginating internally.

7.5 Implications for an MCP-mediated agent

The audit surfaces three concrete recommendations for any agent that wants to filter Zenodo deposits to permissively-licensed, machine-readable artifacts.

1. **Filter on a permissive whitelist, not on the open/closed binary.** The 8-point gap between "open" (85%) and "permissive" (77%) in our sample is the population copyleft and non-commercial restrictions live in. An agent that inherits the user's commercial use case must be told to skip CC-BY-NC and CC-BY-SA variants explicitly.

2. **Treat `[unknown]` and `other-open` as "not yet decided".** Three percent of records have no machine-readable license; ten percent have an umbrella string. Both populations need a follow-up call to `zenodo_get_record` (which returns the full description) and, ideally, a human or a higher-fidelity license-classifier in the loop.

3. **Always filter on `resource_type` after a free-text search.** A bare "machine learning" query returns 45% publications and 30% software; the dataset-bearing 15% is buried. An agent looking specifically for datasets should re-issue with `record_type=dataset` (an authenticated-tier filter) or filter post-hoc on the schema's flattened `resource_type` field.

These three recommendations sit at the wrapper-tools-prompt boundary: they are too specific to bake into a server-side default but too important to leave implicit in the tool docstrings. The natural home for them is the agent's system prompt, alongside the tool descriptions the consuming LLM already reads when picking tools to call.

7.6 Worked example: pinning to a specific dataset version

To make the picture concrete, consider an agent asked to pin a citation to "the version of the Méneame Media Bias Dataset used to train the classifier in this paper":

```
from zenodo_mcp.client import ZenodoClient
from zenodo_mcp.tools import records

c = ZenodoClient()
hits = records.search(c, query="meneame media bias", size=3)
top = hits[0]
record = records.get_record(c, top["record_id"])
```

The result, abridged:

```
{
  "record_id": 19182446,
  "doi": "10.5281/zenodo.19182446",
  "concept_doi": "10.5281/zenodo.19182445",
  "title": "Meneame Media Bias Dataset: Interaction Features and Bias Labels",
  "license": "cc-by-4.0",
  "resource_type": "dataset",
  "publication_date": "2025-...",
  "files": [
    {"key": "...csv", "size": ...,
     "checksum": "md5:abcdef...",
     "download_url": "https://zenodo.org/api/records/19182446/files/.../content
         "},
    ...
  ]
}
```

The record returns a structured `license: cc-by-4.0`, a `resource_type: dataset`, eight files each carrying a download URL and an MD5 checksum, and the concept-DOI alongside the version-DOI. Three calls (search, get-record, optionally get-files) extract everything an agent needs to decide whether the licence permits its intended use, retrieve the artifact, and validate file integrity. A subsequent call to `zenodo_get_versions` returns every version of the dataset under the same concept-DOI, sorted by version, so the agent can pin to a specific revision rather than letting the citation drift.

8 Five wrappers, one composition

`zenodo-mcp` sits within a wider ecosystem of scholarly MCP wrappers. Considered together, peer-review wrappers, integrity wrappers, identity wrappers, venue wrappers, and artifact wrappers compose into something larger than the sum of their parts.

1. Peer-review wrappers surface what reviewers thought of a paper.
2. Integrity wrappers surface whether the paper survived after publication.
3. Identity wrappers surface who the authors actually are, beyond their on-paper names.
4. Venue wrappers surface where the work sits in publishing history.
5. Artifact wrappers (such as `zenodo-mcp`) surface the data, code, and supplementary artifacts the paper describes.

8.1 The composition pattern

A multi-server agent connected to all five families can reason about a paper as a unit of research. A typical composition:

1. Start with a paper DOI from a citation.
2. Resolve the DOI through Crossref to get the title, authors, and venue.
3. Cross-reference each author through an identity wrapper (and optionally a venue wrapper for CS papers) to canonical identifiers.
4. Check the paper through an integrity wrapper for retraction or correction flags.
5. If the paper is at a peer-reviewed venue with open reviews, pull the reviews through a peer-review wrapper.
6. If the paper has supplementary artifacts on Zenodo, pull them through `zenodo-mcp`, filtered on the licence-and-type recommendations of Section 7.

The result is a structured packet of evidence the consuming LLM can reason over: not just "what does this paper say" but "what was its peer-review trajectory, who actually wrote it, did the literature retract any of its claims later, and where are the artifacts" — a fuller picture than any single existing scholarly tool provides.

8.2 What is unique about doing this through MCP

The same composition could in principle be assembled through a hand-written script that calls each upstream API directly. The MCP framing adds three things. First, the agent reasons through the protocol: the LLM picks tools at each step rather than executing a pre-baked script, which lets it skip steps that are not relevant (a humanities paper has no venue-wrapper record; a paper without supplementary materials has no `zenodo-mcp` response). Second, the protocol is composable across other servers: the same agent can interleave `filesystem-mcp` reads, `github-mcp` repository inspection, and `web-mcp` searches. Third, the protocol is portable: the same wrapper works with Claude Desktop, Claude Code, custom Anthropic SDK applications, and any third-party MCP client.

9 Limitations

Five limitations of the v0.1 release deserve explicit mention.

Read-only. The server reads but does not deposit. Many use cases require an agent to *publish* a Zenodo record (a software release, a dataset version). v0.1 deliberately scopes to read-side tools to limit blast radius; a deposit-side extension is on the v0.2 runway under a separate, scope-limited credential.

No bulk-binary streaming. `zenodo_get_files` returns download URLs but does not handle the streaming itself. Consumers fetch the URLs out-of-band, as the MCP transport is not designed for arbitrary binary payloads (Section 6). For very large datasets, Zenodo's bucket API is the appropriate tool, accessible from any HTTP client.

Anonymous-tier page size cap. The public Zenodo API caps anonymous `size` at 25 records per page; authenticated requests can go to 100. The wrapper currently surfaces this only as a 400 error from the upstream; v0.2 will detect the cap and auto-paginate transparently. The same cap affects `zenodo_get_versions`, which currently fails on the anonymous tier for records with at least one extra version (Section 7).

License normalisation. Eleven distinct license strings in a 100-record sample (Section 7) is non-trivial. The wrapper passes the licence string through verbatim; downstream agents see `cc-by` (un-versioned), `cc-by-4.0`, and `cc-zero` as three distinct strings even though one of them is essentially equivalent to another. A normalisation table (e.g. canonicalising `cc-by` to `cc-by-4.0`, or mapping `cc-zero` to `cc0-1.0`) would simplify agent-side filtering at the cost of some loss of fidelity.

Sandbox API not exposed. Zenodo runs a sandbox at `sandbox.zenodo.org` that mirrors the production API for testing. v0.1 does not expose a sandbox switch; deposit-side v0.2 will.

10 Discussion

`zenodo-mcp` closes the artifact-side gap in the academic-MCP catalogue. The strategic motivation is the observation that LLM agents have been treated as *discovery* clients (find me something to read) rather than as *verification and reproduction* clients (does this citation hold up; can I actually run the experiment). Discovery is well-served by existing wrappers; verification and reproduction require integrity checks, canonical identity, and access to artifacts. `zenodo-mcp` contributes the artifact-access piece, with structured-licence and structured-type metadata exposed at exactly the granularity an agent needs to decide whether the artifact is usable.

The 100-record audit (Section 7) is the more interesting half of the contribution. It surfaces three findings that are probably evident to any practising open-data librarian but are not yet evident to the typical LLM agent:

1. License heterogeneity is real. Eleven distinct strings, an 8-point gap between aggregate openness and aggregate permissiveness, three percent of records with no licence at all. An agent that does not filter explicitly will silently violate licence terms.
2. Resource-type heterogeneity is real. A query for "machine learning" returns 45% publications and 30% software; only 15% are datasets. An agent looking for a specific resource type has to filter post-hoc.
3. Public-tier limits leak. Anonymous callers hit a page-size cap that the wrapper does not yet handle gracefully; version-walking fails on the anonymous tier today. This is fixable, and v0.2 will fix it.

We see two natural next directions. First, a deposit-side `zenodo-mcp` would let agents *publish* new records with proper licensing and metadata, which closes a productivity loop (an agent that produces a derived dataset can deposit it under a stable DOI, automatically). Second, a license-normalisation table would compress the eleven-string distribution down to perhaps half a dozen canonical buckets and make agent-side filtering substantially less brittle. Both are low-risk additions; both are on the v0.2 runway.

The thin-wrapper, evidence-first, structured-Pydantic, MCP-transport-for-metadata, out-of-band-fetch-for-bytes pattern is, in our view, the right unit of composition for the next decade of academic agent tooling. We welcome contributions from the community, and we expect the catalogue of academic MCP servers to grow as more under-served scholarly corpora come into reach.

Acknowledgements

We thank CERN and the Zenodo team for operating the open-research infrastructure that makes this server possible.

Availability

Source code, tests, and full API documentation: https://github.com/OpenCodice-Research/zenodo-mcp. License: MIT. Data source: Zenodo REST API. The 100-record audit reported in Section 7 is reproducible with the script `examples/license_audit.py` in the same repository.

References

Anthropic. Model context protocol specification, 2024. URL https://spec.modelcontextprotocol.io.

CERN / OpenAIRE. Zenodo: Open research repository, 2013–present. URL https://zenodo.org/.

Helena Cousijn, Amye Kenall, Emma Ganley, Melissa Harrison, David Kernohan, Thomas Lemberger, Fiona Murphy, Patrick Polischuk, Simone Taylor, Maryann Martone, and Timothy Clark. A data citation roadmap for scientific publishers. *Scientific Data*, 5:180259, 2018. .

Janne Neumann and Jan Brase. DataCite and DOI names for research data. *Journal of Computer-Aided Molecular Design*, 28(10):1035–1041, 2014. .

Irene V. Pasquetto, Bernadette M. Randles, and Christine L. Borgman. On the reuse of scientific data. *Data Science Journal*, 16:8, 2017. .

Arfon M. Smith, Daniel S. Katz, and Kyle E. Niemeyer. Software citation principles. *PeerJ Computer Science*, 2:e86, 2016. .

Shelley Stall, Lynn Yarmey, Joel Cutcher-Gershenfeld, Brooks Hanson, Kerstin Lehnert, Brian Nosek, Mark Parsons, Erin Robinson, and Lesley Wyborn. Make scientific data FAIR. *Nature*, 570(7759):27–29, 2019. .

Nicole A. Vasilevsky, Jessica Minnier, Melissa A. Haendel, and Robin E. Champieux. Reproducible and reusable research: are journal data sharing policies meeting the mark? *PeerJ*, 5:e3208, 2017. .

Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, et al. The FAIR guiding principles for scientific data management and stewardship. *Scientific Data*, 3:160018, 2016. .

OpenCódice Technical Report OC-TR-2026-011 • 2026 • OpenCódice Research
opencodice.org • DOI: 10.5281/zenodo.20023539