OpenCódice Technical Report OC-TR-2026-007

openreview-mcp: A Model Context Protocol Server for Querying Peer Review at Scale

Francisco-Javier Rodrigo-Ginés
NLP & IR Group, UNED
fran@opencodice.org

April 25, 2026 • **DOI:** 10.5281/zenodo.19758460

Abstract

The Model Context Protocol (MCP) ecosystem now covers arXiv, Semantic Scholar, Hugging Face, and Crossref, but not peer review: the reasoning of expert reviewers, hosted publicly on OpenReview, is unreachable by any MCP-connected LLM. This report introduces `openreview-mcp`, an open-source MCP server that closes the gap. It exposes eleven tools across four entity families plus a signature analysis tool, `openreview_aggregate_weaknesses`, which clusters recurrent reviewer complaints and returns raw evidence (top terms, exemplars, submission ids) for the consuming agent to label. A reproducible case study on 100 rejected ICLR 2024 submissions yields 1,361 weakness items across 14 clusters and surfaces a non-obvious finding: evaluation concerns drive 42% of complaints, while generic novelty complaints account for less than 8%, contradicting the folklore that novelty is the primary battleground in machine-learning peer review.

Keywords: Model Context Protocol, peer review, OpenReview, research tooling, agentic research, text clustering, reviewer modelling

Highlights

- Open-source MCP server exposing 11 OpenReview tools (venues, submissions, reviews, decisions, profiles, analysis)
- Signature tool `openreview_aggregate_weaknesses`: clusters recurrent reviewer complaints with TF-IDF, LSA, and KMeans
- Returns raw evidence (top terms, exemplars, submission ids), letting the consuming LLM label clusters appropriately to each venue
- Reproducible ICLR 2024 case study: 1,361 weaknesses from 100 rejections, 14 clusters, headline finding contradicts the novelty folklore

1 Introduction

The Model Context Protocol [Anthropic, 2024] has matured rapidly into the standard interface between large language models and external systems. By the time of writing, the public registry of MCP servers covers most of the everyday tooling stack a researcher relies on: 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 in this ecosystem concerns peer review itself. Most academic-search MCP servers treat papers as the unit of information: title, abstract, citations, PDF link. This framing is sufficient for discovery but ignores what is arguably the densest signal academic machine learning produces, namely the reasoning of expert reviewers explaining why a paper succeeds or fails. That reasoning is hosted publicly, in structured form, on OpenReview [Soergel et al., 2013] for nearly every major machine-learning venue: ICLR, NeurIPS, COLM, TMLR, ACL ARR, and dozens of workshops. Reviews, author rebuttals, area-chair meta-reviews, and final decisions are queryable through an official API. They are simply not reachable by any LLM connected through MCP.

This report introduces `openreview-mcp`, an open-source MCP server that closes the gap. We make three contributions.

- **A complete MCP surface for OpenReview.** Eleven tools cover the four major entity families (venues, submissions, reviews, profiles), preserve the protocol's two-API-version structure (v1 legacy and v2 current), and ship as a standard Python package installable with `pip`.
- **A signature analysis tool, `openreview_aggregate_weaknesses`.** It samples rejected submissions from a venue, splits each reviewer's weakness paragraph into individual items, vectorizes with TF-IDF, reduces with Truncated SVD, and clusters with KMeans. Unlike pre-baked taxonomies, it returns raw evidence (top terms and nearest-to-centroid exemplars) and lets the consuming LLM produce labels appropriate to whatever venue and slice of literature it is studying.
- **A reproducible case study on ICLR 2024.** 100 rejected submissions yielded 1,361 weakness items clustered into 14 themes. The headline finding (Section 6) is non-obvious: evaluation concerns dominate; generic novelty complaints are the fifth-largest cluster, not the first.

The server is released under the MIT license at `https://github.com/OpenCodice-Research/openreview-mcp` and is installable as `pip install openreview-mcp[analysis]`. It is the first MCP server published by OpenCódice Research as part of a planned series targeting under-served corpora in the academic-research stack.

The remainder of this report is organised as follows. Section 2 reviews the Model Context Protocol, the existing academic MCP server landscape, and the OpenReview platform. Section 3 documents the system architecture. Section 4 catalogues the eleven tools and the schema design. Section 5 describes the analysis pipeline in detail and the design rationale behind returning raw evidence. Section 6 presents the ICLR 2024 case study. Sections 7 and 8 discuss use cases, limitations, and future work.

2 Background

This section sketches the three contextual elements the rest of the report builds on: the Model Context Protocol as the integration substrate, the existing landscape of academic MCP servers, and the OpenReview platform as a queryable corpus.

2.1 The Model Context Protocol

The Model Context Protocol is an open specification [Anthropic, 2024] that lets LLM applications expose external capabilities (tools, resources, prompts) to language models through a uniform interface. An MCP *server* declares its tools with JSON schemas; an MCP *client* (such as Claude Desktop, Claude Code, or a custom application) connects to the server, lists its tools, and lets the underlying language model invoke them as part of a conversation. Two transports are standard: `stdio` for locally-spawned subprocesses and `streamable-http` for network-attached services.

For academic workflows, the value proposition is composability. A researcher can connect a single LLM session to multiple domain servers (literature search, code execution, dataset access, peer review) and let the model orchestrate them as it reasons. The protocol is rapidly becoming the default integration substrate for research-assistant agents.

2.2 Existing academic MCP servers

Several academic MCP servers already exist. The most established is Academia MCP [Gusev, 2025], which exposes arXiv search and download, ACL Anthology search, Semantic Scholar citation graphs, and Hugging Face dataset discovery, plus utilities for LaTeX template enumeration. Smaller servers cover specific subsystems: arXiv-only wrappers, PubMed bridges, Crossref DOI resolution, and Scholar parsers. None of these expose peer review.

The omission is not accidental. Peer review on OpenReview is structurally heterogeneous: invitation patterns, content schemas, and review templates vary across venues and across years within a venue. Building a robust integration requires handling that heterogeneity at the wrapper layer rather than the agent layer. The contribution of `openreview-mcp` is precisely that handling.

2.3 OpenReview as a corpus

OpenReview [Soergel et al., 2013] hosts open peer review for venues that opt into transparency. Submissions, reviews, rebuttals, area-chair meta-reviews, and final decisions are stored as *notes* linked to a common *forum*. The platform exposes two API generations in parallel: v1 (legacy invitations, used by most pre-2023 venues) and v2 (current, used by all new venues). The v2 API wraps every content field as `{value: ...}`, which v1 does not, requiring a normalisation layer.

The corpus is large and growing. ICLR 2024 alone hosts 7,404 submissions; NeurIPS 2024 hosts more. Reviews are typically multi-paragraph, with explicit subsections for summary, strengths, weaknesses, questions, and limitations. Many venues additionally collect numeric ratings for soundness, presentation, and contribution. This structural richness is what makes OpenReview a uniquely tractable corpus for downstream analysis: the unit of analysis can be a submission, a review, a single complaint within a review, or an aggregate across thousands of reviews.

3 System architecture

`openreview-mcp` is a small Python package built around four concerns: a thin client wrapper over the official OpenReview SDK, 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

The runtime stack is the following, from bottom to top:

1. **Client wrapper** (`openreview_mcp.client`). Lazy-instantiates v1 and v2 OpenReview SDK clients. Reads optional credentials from `OPENREVIEW_USERNAME` and `OPENREVIEW_PASSWORD`; defaults to anonymous read-only access for public venues.

2. **Cache** (`openreview_mcp.cache`). Diskcache-backed key-value store with per-tool TTLs (six hours to seven days). Cache is bypassed when `OPENREVIEW_MCP_NO_CACHE=1`. The decorator

preserves return-type information through `ParamSpec` and `TypeVar` so static type checkers track tool signatures correctly.

3. **Schemas** (`openreview_mcp.schemas`). Pydantic v2 models (`Venue`, `Submission`, `Review`, `MetaReview`, `Rebuttal`, `Decision`, `Profile`, `WeaknessCluster`, `WeaknessAggregate`). Validation catches API drift early.
4. **Tools** (`openreview_mcp.tools`). Pure functions grouped by family: `venues.py`, `submissions.py`, `reviews.py`, `profiles.py`, `analysis.py`. Each function takes the client as its first positional argument plus typed kwargs and returns validated dictionaries.
5. **Server** (`openreview_mcp.server`). A FastMCP application that wraps each tool function with an `@mcp.tool()` decorator and a docstring suitable for LLM consumption.
6. **Transport** (`openreview_mcp.cli`). Argparse-based CLI selects between `stdio` (default, for locally-spawned MCP clients) and `streamable-http` (for network-attached deployments).

3.2 Content normalisation

OpenReview's two API generations differ in how they encode content fields: v2 wraps every field as `{"value": ...}`; v1 stores raw values. The wrapper layer hides this through a small helper, `content_value`, that unwraps automatically. A second helper, `as_str_list`, normalises fields that may be either string-valued (e.g. free-form ethics flags) or list-valued (e.g. multi-flag ethics responses). These helpers were both prompted by real validation errors hitting the schema layer during initial integration testing against ICLR 2024 data.

3.3 Caching strategy

Most OpenReview data is effectively immutable: once a venue's review period closes, reviews and decisions do not change. We cache aggressively (TTL 24 hours for reviews and decisions, seven days for analysis aggregates) and key on a stable hash of the function name and arguments. Since the consuming agent typically issues many small queries against the same venue (e.g. list submissions, then iterate over reviews), the cache hit rate is high in practice.

3.4 Failure semantics

Tools fail loudly. Authentication errors, missing venue ids, and SDK exceptions propagate to the MCP client as tool errors with a human-readable message. The aggregate analysis tool raises a clear `RuntimeError` when the optional `[analysis]` extra (scikit-learn, NumPy) is not installed, telling the user how to install it.

4 Tools

The server exposes eleven tools grouped into four entity families plus an analysis family. Table 1 summarises the surface.

4.1 Schema design

Every tool returns a Pydantic-validated dictionary. Key choices:

- **Flat output, no nested objects.** Tools return shallow dictionaries; cross-entity links use ids (`forum`, `submission_id`, `author_id`). This keeps the LLM-facing surface predictable.
- **Optional fields default to `None`, lists default to empty.** OpenReview reviews are heterogeneous across venues; soundness and presentation are NeurIPS conventions, summary and weaknesses are ICLR conventions, etc. Schemas accept all of them; missing fields are simply `None`.

Family	Tool	Purpose
Venues	`list_venues`	List venues filtered by year or series
	`venue_stats`	Submission count, accept rate, rating histogram
Submissions	`search_submissions`	Search by venue, query, author, keywords
	`get_submission`	Fetch full submission metadata
	`search_by_author`	List a profile's submissions
Reviews	`get_reviews`	Pull all official reviews with scores
	`get_meta_review`	Area-chair meta-review
	`get_rebuttal`	Author rebuttals keyed to reviews
	`get_decision`	Final accept/reject decision
Profiles	`get_profile`	Resolve names, affiliations, identifiers
Analysis	`aggregate_weaknesses`	Cluster recurrent reviewer complaints

Table 1: The eleven tools exposed by `openreview-mcp`, grouped by entity family. All tools are prefixed with `openreview_` when registered as MCP tools.

- **Numeric coercion for ratings.** Reviewers commonly submit ratings as strings like `"7: good"`; a small helper extracts the leading numeric portion as a float.
- **Anonymised reviewer ids.** The first signature on a review is returned as `reviewer_id`; for ICLR/NeurIPS this is a venue-anonymous handle (e.g. `Reviewer_aaa1`), preserving consistency across reviews from the same person within a forum without revealing identity.

4.2 Authentication

The wrapper supports anonymous access for public venues (the common case) and credential-based access for private venues, ARR cycles before public release, or workshop sessions with restricted visibility. Credentials are read from environment variables and never persisted. No secret storage is required.

5 The aggregate-weaknesses pipeline

The signature analysis tool, `openreview_aggregate_weaknesses`, is what differentiates the server from a thin REST wrapper. It surfaces structural patterns across thousands of reviewer complaints in a way that no per-paper query can.

5.1 Design rationale

We considered three candidate architectures.

1. **Pre-baked taxonomy.** Define a fixed set of weakness categories (*novelty*, *experiments*, *writing*, etc.) and use a lightweight classifier to assign reviews to categories. Rejected: categories vary across venues and across years; freezing them inside the server creates drift the moment a community evolves its norms.
2. **LLM-only summarisation.** Have the consuming LLM read all weakness paragraphs and write a free-text summary. Rejected: token costs scale linearly with sample size; the same query repeated yields different answers; the analysis is not reproducible.
3. **Deterministic clustering with raw evidence return.** Cluster on the server with a fixed seed; return top terms, exemplars, and contributing submission ids; let the consuming LLM produce labels from the evidence. **Selected.**

The selected design has three useful properties. The clustering is reproducible (fixed seed, deterministic algorithm). The labels are flexible (the LLM picks vocabulary appropriate to the

venue and audience). The evidence is auditable (every cluster cites three exemplar quotes and a list of submission ids, which can be reopened with `get_submission`).

5.2 Pipeline

The pipeline executes in seven steps, summarised in Algorithm 1.

Algorithm 1 `aggregate_weaknesses`

Require: Venue id V, sample size n, target cluster count k, seed s
Ensure: Aggregate of k clusters with top terms, exemplars, submission ids

1: Fetch all submissions in V with embedded replies (one bulk API call)
2: Filter to submissions with decision containing "`reject`"
3: Sample n submissions with seed s
4: For each submission, extract `weaknesses` from every Official_Review reply
5: Split each paragraph into individual items on bullet/numbered delimiters
6: Vectorise items with TF-IDF (sublinear TF, 1–2 grams, $\min_{df} = 3$, $\max_{df} = 0.5$)
7: Reduce to 64 latent dimensions with Truncated SVD; re-normalise
8: Cluster with KMeans (k, $n_{init} = 20$)
9: For each cluster: compute top TF-IDF terms over members, select 3 nearest-to-centroid exemplars, list contributing submission ids
10: Sort clusters by size, return as `WeaknessAggregate`

5.3 Item splitting

A reviewer's `weaknesses` field typically contains five to eight distinct complaints, separated by markers like "`- `", "`1. `", "`(1) `", "`W1.`", or "`##`". Clustering at the level of full paragraphs collapses into one giant cluster (we observed 90% of items absorbed by a single centroid in early experiments) because the per-review texts share too much vocabulary. Splitting to per-complaint items dramatically improves separability: each item targets a single concern, so vocabulary co-occurrence becomes informative again. The default behaviour can be disabled with `split_items=False` if a coarser, per-review clustering is desired.

5.4 TF-IDF + LSA + KMeans

Plain KMeans on sparse TF-IDF vectors with a small target k performs poorly on review text: the bag-of-words representation has too much shared vocabulary across the long tail of complaints, and Euclidean centroid updates collapse into one dominant cluster. We adopt the classical text-clustering pipeline: TF-IDF, then dimensionality reduction with Truncated SVD to 64 latent dimensions (the LSA approach of Deerwester et al., 1990), then L_2 re-normalisation, then KMeans with twenty random initialisations. Top terms per cluster are computed by averaging TF-IDF rows over cluster members, since LSA-space centroids do not map cleanly back to term weights.

This pipeline produces well-balanced cluster sizes and interpretable top-term lists across venues we have tested. It runs in seconds on samples of up to several thousand items and adds no per-query inference cost beyond a bulk API fetch.

6 Case study: ICLR 2024

To validate the pipeline we point it at a venue many readers know first-hand: ICLR 2024.

6.1 Setup

```
from openreview_mcp.client import OpenReviewClient
from openreview_mcp.tools import analysis
```

```
client = OpenReviewClient()
result = analysis.aggregate_weaknesses(
    client,
    venue_id="ICLR.cc/2024/Conference",
    sample_size=100,
    n_clusters=14,
    min_text_length=40,
    seed=7,
)
```

The sample draws from the 7,404 ICLR 2024 submissions filtered to those with a `reject` decision. Bulk fetch with embedded replies takes approximately thirty seconds; clustering completes in under one second.

6.2 Results

The 100 sampled submissions yielded 1,361 individual weakness items and fourteen clusters, summarised in Table 2.

Cluster	Items	Papers	Top terms (top 6)
13	197	79	evaluation, tasks, work, performance, using, model
2	194	66	models, equation, model, does, assumption, information
1	180	73	experiments, authors, results, datasets, data, experimental
8	123	61	method, proposed, proposed method, does, paper, unclear
9	104	55	paper, lacks, paper lacks, main, weaknesses, authors
3	88	43	learning, federated, federated learning, arxiv, supervised, algorithm
11	83	51	methods, existing, based, authors, proposed, method
10	78	40	section, authors, work, clarity, does, paper
6	71	42	et al, al, et, learning, 2022, 2023
7	68	32	loss, eq, function, loss function, notation, defined
4	50	38	figure, figures, clarity, fig, captions, memory
5	50	22	time, time series, series, authors, complexity, step
0	44	34	writing, presentation, improved, example, paper, readability
12	31	11	https, org, html, fairness, com, pdf

Table 2: Cluster output of `aggregate_weaknesses` on 100 rejected ICLR 2024 submissions ($n_{\text{items}} = 1{,}361$, $k = 14$, seed $= 7$). Cluster ids are arbitrary; clusters are ordered by size. Cluster 12 (URLs extracted from reviews) is a noise cluster.

The labels in Table 2 are produced by reading the top terms and the exemplar quotes returned by the tool, exactly the labelling task the consuming LLM performs in normal use.

6.3 Findings

Three observations stand out.

Evaluation, not novelty, drives most rejections. The three largest clusters (13, 2, 1) total 571 items, or 42% of the sample. Cluster 13 collects complaints about evaluation that is too narrow or too biased; cluster 2 collects modelling and assumption opacity; cluster 1 collects shallow experiments and weak theoretical results. The generic "paper lacks X" cluster (9), often the canonical novelty critique, is only the fifth-largest with 104 items. Reviewers do raise novelty concerns, but they raise evaluation concerns nearly four times as often.

Craftsmanship still sinks papers. Typos and broken cross-references (cluster 10, 78 items), confusing figures and captions (cluster 4, 50 items), missing citations (cluster 6, 71 items), and explicit complaints about writing quality (cluster 0, 44 items) together flag roughly 70 of the 100 sampled papers. These are mechanical issues that a careful proof-reading pass would catch, and yet they are routine drivers of explicit reviewer complaints in rejected submissions.

Topic-specific failure modes emerge. Two clusters are narrowly topical: federated and semi-supervised learning (cluster 3, 88 items) and time series (cluster 5, 50 items). These likely reflect both high submission volume in those areas and domain-specific failure modes, such as unrealistic non-i.i.d. data settings for federated work. Authors submitting in those areas would do well to read the cluster exemplars and pre-rebut the standard objections.

6.4 An exemplar

A single complaint extracted from cluster 11 (insufficient comparisons) illustrates the level of detail the tool surfaces:

> **Reviewer (ICLR 2024, rejected submission, cluster 11).** "Though the authors claim that they aim to propose a unified framework, the methods considered in their paper are mainly based on AM and POMO, in other words, the auto-regressive methods. As far as I know, there are also other methods (. . .)."

This is what `aggregate_weaknesses` returns as evidence: the exact language and detail that rejected papers faced, not abstract category labels.

6.5 Reproducibility

The exact invocation, the full cluster table, the per-cluster top terms, and three exemplar quotes per cluster are stored at https://github.com/OpenCodice-Research/openreview-mcp/blob/main/examples/iclr2024_case_study.md. The diskcache layer makes the result deterministic across runs as long as the OpenReview API state is unchanged; deleting the cache and re-running with the same seed reproduces the same fourteen clusters.

7 Discussion

The system and case study described above are the technical core of this report. The remaining question is what they are good for, where they fall short, and how they compare to the alternatives. This section discusses the four use cases we anticipate, the limitations practitioners should keep in mind, and the cost-latency profile of the tool in practice.

7.1 Use cases

We anticipate four primary use cases for `openreview-mcp`.

> **Pre-submission self-review.** Before submitting to a venue, an author can run `aggregate_weaknesses` on that venue's recent rejections and ask the consuming LLM which clusters their draft is most exposed to. The LLM can then read the exemplar quotes and produce a targeted "what reviewers will hate" checklist that grounds critique in the venue's actual reviewer language rather than a generic rubric.

Reviewer-style critique agents. Tools that simulate harsh reviewers often suffer from the "everything is fine in the abstract, everything is wrong in detail" failure mode because they have no anchor in real reviewer behaviour. Pairing such an agent with `get_reviews` on similar accepted/rejected submissions and with `aggregate_weaknesses` on the target venue grounds the critique in observable patterns.

Teaching. PhD advisors routinely tell students what venues want and what they reject. The advice is usually correct in spirit but biased by the advisor's own submission history. With `openreview-mcp` a student can construct an evidence-based view of the venue from the actual review record, complementing the advisor's narrative with reproducible aggregates.

Rebuttal mining. Pairing `get_rebuttal` with `get_decision` enables a different style of analysis: among borderline submissions where the rebuttal flipped the outcome, what rhetorical patterns recurred? Future tools could automate this comparison; for now the primitives are available.

7.2 Limitations

Sample size and venue specificity. The case study draws 100 submissions from one venue in one year. Patterns may not generalise across venues, across years within a venue, or across rejection bands within a year. Practitioners should run the tool on their actual target venue rather than assume the ICLR 2024 patterns hold.

Lexical clustering. TF-IDF captures lexical similarity, not semantic similarity. Reviews that complain about the same underlying issue using different vocabulary may end up in different clusters. Sentence-embedding clustering with a model like `sentence-transformers` would likely surface finer structure at the cost of an additional dependency and a slower runtime; we plan to support it as an opt-in alternative pipeline.

Sampling uniformity. Submissions are sampled uniformly across rejected papers. Borderline rejections (decision text *Reject (close)*) carry different complaint patterns than clear rejections; we currently treat them identically. Weighting by reviewer confidence or decision margin would bias toward stronger signals at the cost of reduced coverage.

Noise clusters. The pipeline does not filter URL-only or citation-only items, which form their own small cluster (12 in the case study). The consuming LLM can identify and ignore them, but a future improvement is to filter such items before vectorisation.

Privacy. The server only exposes data already public on OpenReview. No additional information is leaked. Users querying private venues with credentials are responsible for respecting the access constraints of those venues.

7.3 Comparison to related approaches

Compared to manual qualitative coding of reviews, `aggregate_weaknesses` trades nuance for scale: a human coder produces deeper categories on a smaller sample; the tool produces shallower clusters on a larger sample, fast enough to re-run as new venues close. Compared to fully LLM-driven summarisation, the tool trades adaptability for reproducibility: the LLM can summarise any sample, but the same query yields different answers across runs and incurs significant token cost. The clustering pipeline is deterministic and incurs zero LLM cost; the LLM is engaged only at the labelling step, where its variability is a feature rather than a bug.

7.4 Cost and latency

Bulk fetching all submissions from a venue with embedded replies takes 20–40 seconds for a venue of ICLR/NeurIPS scale; the result is cached for seven days. Clustering on ∼1,500 items completes in under one second. The MCP tool round-trip from a client like Claude Code is dominated by the bulk fetch on first call and is essentially free on cache hits. Token cost on the LLM side depends only on the size of the returned aggregate, which is fixed by k and the exemplar quote length, not by the sample size.

8 Conclusion and future work

`openreview-mcp` closes the largest remaining gap in the academic Model Context Protocol ecosystem. The server exposes peer-review reasoning, the densest signal academic machine learning produces, as eleven composable tools any MCP-compatible LLM can call. A signature analysis tool, `aggregate_weaknesses`, surfaces structural patterns across thousands of reviewer complaints with a deterministic clustering pipeline that returns raw evidence and lets the consuming LLM produce labels.

The ICLR 2024 case study demonstrates the value of the approach: 100 rejected submissions yield 1,361 weakness items in 14 coherent clusters, leading to an evidence-grounded headline finding (evaluation, not novelty, drives most rejections) that contradicts widespread folklore about machine-learning peer review. The cluster table and exemplar quotes are reproducible from the same seed.

Future work. Three directions are immediate. First, an optional sentence-embedding pipeline (`sentence-transformers`) for finer-grained semantic clustering, gated as an additional install extra. Second, venue-specific normalisation profiles for venues with non-standard review templates (TMLR's continuous review, ARR's cycle-based reviews). Third, a public dashboard refreshed weekly that runs `aggregate_weaknesses` against every major venue as soon as its decisions go live, to track how reviewer patterns evolve.

Availability. The server is published on PyPI as `openreview-mcp` (with `[analysis]` extra for the clustering pipeline) and at https://github.com/OpenCodice-Research/openreview-mcp under the MIT license. Issues, contributions, and venue-specific bug reports are welcome. The repository is the first in a planned series of MCP servers from OpenCódice Research targeting under-served corpora in the academic-research stack.

References

Anthropic. Introducing the Model Context Protocol. https://www.anthropic.com/news/model-context-protocol, 2024. Specification at https://modelcontextprotocol.io.

Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990. .

Ilya Gusev. Academia MCP: Tools for automatic scientific research. https://github.com/IlyaGusev/academia_mcp, 2025. MCP server exposing arXiv, ACL Anthology, Semantic Scholar and Hugging Face datasets.

David Soergel, Adam Saunders, and Andrew McCallum. Open scholarship and peer review: a time for experimentation. ICML Workshop on Peer Reviewing and Publishing Models, 2013. Foundational vision for the OpenReview platform; see https://openreview.net.

OpenCódice Technical Report OC-TR-2026-007 • 2026 • OpenCódice Research

opencodice.org • DOI: 10.5281/zenodo.19758460