Google Scholar API — Technical Report

Author: Yessamin Andrade
Date: October 1st, 2025
Repository: https://github.com/yessaminandrade/Server_Database#

1. *Introduction*
   1.1 Purpose
   • Summarize how to access and use Google Scholar data via SerpApi for literature queries and metadata extraction.
   • Scope: endpoints, authentication, query parameters, response formats, usage limits, and multi-language examples.

1 References
https://serpapi.com/search-api

2. Endpoints (URLs for API functions)
   2.1 Base Endpoint
   • Base URL (JSON): https://serpapi.com/search.json
   • Protocol: HTTPS
   • Default output format: JSON

2.2 Endpoint Matrix

• Sample rows:
– google_scholar (Articles) | GET | /search.json | Article search | engine=google_scholar&q=<...>
– google_scholar_profiles (Author Profiles) | GET | /search.json | Author profile search | engine=google_scholar_profiles&mauthors=<...>

Note: Behavior is selected via the engine parameter.

3. *Authentication Methods (How to obtain and use keys/tokens)*
   3.1 Obtaining the API Key
   • Create a SerpApi account (free tier).
   • Copy the API key from your dashboard.

3.2 Including the Key in Requests
• Send as a query parameter: api_key=<YOUR_KEY>.

3.3 Security Best Practices
• Do not expose the key in public client-side code.
• Add ".env" to .gitignore and provide a sanitized ".env.example".

4. *Query Parameters (Filters and customization)*
   4.1 Common Parameters
   • Sample rows:
   – engine | string | Yes | google_scholar / google_scholar_profiles | Selects data source/behavior

– api_key | string | Yes | xxxx | SerpApi API key
– hl | string | No | en / es | Interface language
– num | int | No | 10 | Results per page
– start | int | No | 0 | Pagination offset

4.2 Article Search (engine=google_scholar)

• Sample rows:
– q | string | Yes | deep+learning | Search query
– as_ylo | int | No | 2019 | Year from (inclusive)
– as_yhi | int | No | 2025 | Year to (inclusive)
– (Add others as confirmed in official docs)

4.3 Author Profiles (engine=google_scholar_profiles)

• Sample row:
– mauthors | string | Yes | Yoshua+Bengio | Author name to search

5. *Response Formats (How data is structured)*
   5.1 Articles — Typical Fields (engine=google_scholar)
   • search_metadata, search_parameters
   • organic_results[]:
   – title, link, snippet
   – publication_info (authors, venue, year)
   – resources[] (e.g., PDF links)
   – inline_links.cited_by.total (when present)
   • serpapi_pagination

5.2 Author Profiles — Typical Fields (engine=google_scholar_profiles)
• search_metadata, search_parameters
• profiles[]:
– name, affiliations, link, cited_by, thumbnail

5.3 Example Excerpts
• Insert a short, trimmed JSON excerpt (1–3 items) as plain text or a figure.
• Optionally reference saved sample responses in a “samples/” folder.

6. Usage Limits (Request restrictions)
   6.1 Quotas and Rate Limits
   • Free tier includes a limited number of requests per period (verify your current quota in the dashboard).
   • Exceeding limits may return “429 Too Many Requests.”

6.2 Common Errors and Handling
(Insert a Word table)

• Columns: Status Code | Cause | Recommended Action

• Sample rows:
– 401 | Missing/invalid api_key | Verify key; include as query parameter
– 422 | Invalid or missing parameters | Validate required parameters
– 429 | Rate limit or quota exceeded | Implement backoff/retry; check quota

7. *Code Examples (Multi-language demonstrations)*

7.1 cURL

• One-line request for article search with q, num, hl, and api_key.

"https://serpapi.com/search.json?engine=google_scholar&q=<query>&num=5&api_key=$SERPAPI_KEY"

7.2 Python (requests)

```
import os, requests

params = {

  "engine": "google_scholar",

  "q": "deep learning",

  "num": 5,

  "api_key": os.getenv("SERPAPI_KEY")

}

r = requests.get("https://serpapi.com/search.json", params=params, timeout=30)

r.raise_for_status()

data = r.json()

print(data.get("organic_results", [])[0].get("title"))
```

7.3 Java (OkHttp or HttpClient)

```
OkHttpClient client = new OkHttpClient();

HttpUrl url = HttpUrl.parse("https://serpapi.com/search.json").newBuilder()

  .addQueryParameter("engine", "google_scholar")

  .addQueryParameter("q", "graph neural networks")

  .addQueryParameter("num", "5")

  .addQueryParameter("api_key", System.getenv("SERPAPI_KEY"))

  .build();

Request req = new Request.Builder().url(url).build();

try (Response res = client.newCall(req).execute()) {

  if (!res.isSuccessful()) throw new IOException("HTTP " + res.code());
```

```
  System.out.println(res.body().string());
}
```

8. *Conclusions*

   The SerpApi implementation for Google Scholar provides a practical, reliable way to programmatically access **article results** and **author profiles** via a single JSON endpoint (/search.json) with behavior selected through the engine parameter.

9. **Authentication** is straightforward (API key in query), but must be handled securely (environment variables, .env, never committed) to prevent exposure in public code.

10. The **query model** is simple yet flexible: common controls (hl, num, start) plus engine-specific filters (e.g., q, as_ylo, as_yhi for articles; mauthors for profiles) support focused searches and reproducible queries.

11. **Response structures** are consistent and easy to parse: organic_results[] for articles and profiles[] for author searches, alongside search_metadata, search_parameters, and pagination details. This enables quick extraction of titles, links, citations, authors, and venues.

12. **Usage limits** on the free tier are sufficient for exploration and small pipelines; rate limiting and common error codes (401, 422, 429) are predictable and can be mitigated with basic retry/backoff and parameter validation.

13. The API is **language-agnostic** and easy to adopt: short examples in cURL, Python, Node.js, and Java demonstrate how to authenticate, request results, and read key fields with minimal setup.

14. For coursework and lightweight research tasks, this approach delivers **fast time-to-value**—you can document endpoints, test queries, store sample JSON, and integrate outputs into data workflows with very little overhead.

15. The main **trade-off** is dependence on a third-party aggregator (SerpApi) and its quotas; for high-volume or production-critical scenarios, plan for quota management, caching, and graceful degradation.