Google Summer of Code 2026 - OSGeo Application

JSON Handling Toolkit for the QGIS Processing Framework

Sionigdha Sadhukhan | snigdha.lee75@gmail.com | github.com/Sionigdha

1. Contact Details

Name and Surname: Sionigdha Sadhukhan
Nickname: Sionigdha
Country: India
Email: snigdha.lee75@gmail.com
Phone: +91-8420871910
Public repositories:
https://github.com/Sionigdha
https://github.com/OSGeo/gdal/pulls?q=author%3ASionigdha|
https://github.com/qgis/QGIS/pulls?q=author%3ASionigdha
LinkedIn: www.linkedin.com/in/sionigdha-sadhukhan-564b48218
Timezone: IST (UTC+5:30)

2. The Idea

OSGeo member software: QGIS
Title: JSON Handling Toolkit for the QGIS Processing Framework

Brief Description

My project will focus on adding JSON as a proper data type in the QGIS processing framework. Right now there is no built-in way to load a JSON file into a model, pull a value out of a JSON string, or convert a JSON array into a vector layer, so you have to write a custom Python script for all of it. I want to fix that with four new processing algorithms: LoadJSONFile, ExtractJSONValue, FlattenJSON, and JSONToTable. I will also I will extend the existing ogrinfo and ogrinfojson GDAL processing algorithms with typed parameters that are currently only reachable via a raw string field. This direction was suggested and validated by Valentin Buira on the QGIS developer mailing list in March 2026.

State of the Software Before GSoC

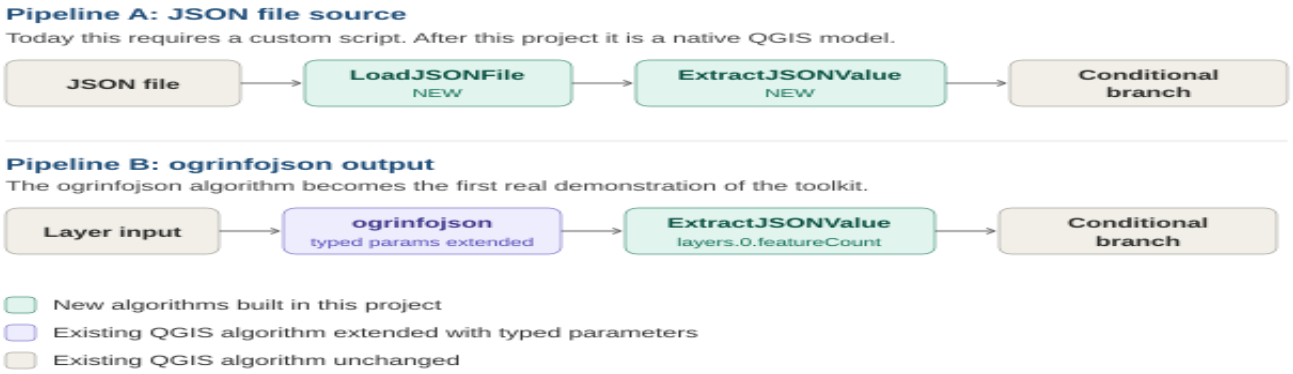
In QGIS today, if I run ogrinfojson on a vector layer and want to use the feature count from its output to drive a conditional branch in a model, I cannot do it directly. The algorithm writes a JSON file, but no built-in algorithm can open that file and extract a value from it. I have to write a custom Script algorithm, manually call json.load(), traverse the dictionary to get layers[0]['featureCount'], and return it. That script is mine alone, not reusable, and not visible as a proper node in the model. This same gap appears whenever JSON data enters a QGIS workflow from any source: REST APIs, WFS metadata, plugin outputs. The ogrinfojson case is just one instance of a framework-wide missing capability. There is simply no JSON support in the processing framework.

On the ogrinfo side specifically: the algorithms have an EXTRA parameter where users can manually type raw GDAL flag strings like -where "name='road'" or -fid 42. This technically works but there is no type checking, no UI widget, no validation, and it is not obvious to anyone looking at the algorithm that these options exist. Seven useful flags: -where, -fid, -sql, -spat, -geom, -fields, -limit. are not exposed as proper typed parameters.

What My Project Will Bring

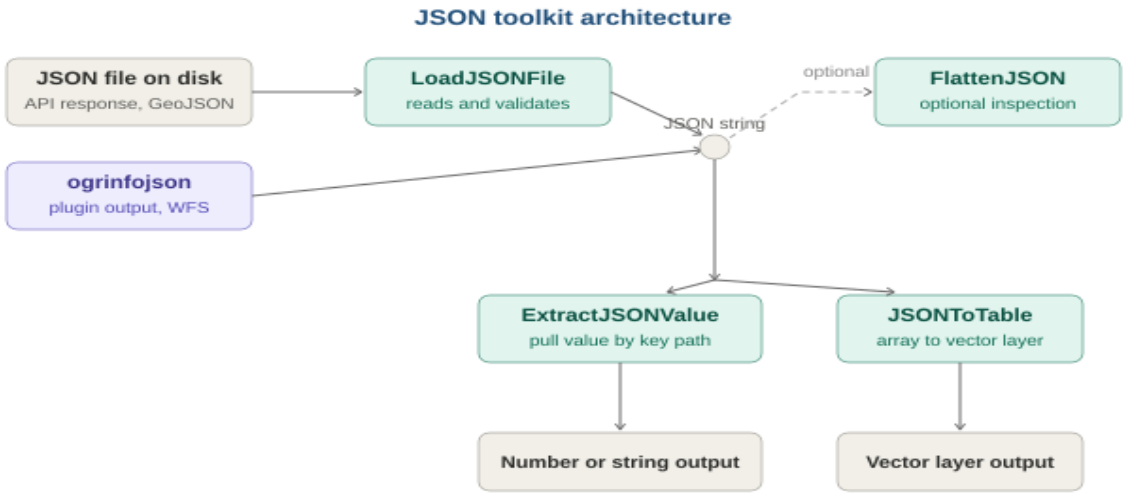
After this project, the same ogrinfojson workflow becomes a clean three-step model: run ogrinfojson on a layer, connect its output directly to ExtractJSONValue with the key path layers.0.featureCount, and feed the result into a conditional branch. No custom script needed. No manual JSON parsing. The whole thing becomes a standard QGIS model that anyone can open, read, and reuse.

More broadly, the four new algorithms will make it possible to:



- Load any JSON file into a model as a typed input (LoadJSONFile)
- Extract any value from a JSON string by key path (ExtractJSONValue)
- Inspect the full structure of an unknown JSON document (FlattenJSON)
- Convert a JSON array directly to a vector layer with proper attribute fields (JSONToTable)

On the ogrinfo side, all seven flags will become proper typed parameters: a string field with validation for -where, a number spinner for -fid, an extent picker for -spat, dropdown enums for -geom and -fields. The EXTRA parameter stays for backwards compatibility but gets marked as deprecated.



Future Developments

The most natural next step after this project is a dedicated QgsProcessingParameterJSON type. Right now JSON strings pass between algorithms as QgsProcessingOutputString, which works but does not give the model designer any way to know the output is specifically JSON. A proper JSON parameter type would enable validation, autocomplete for key paths, and better error messages. That is a C++ core change that is out of scope for this project but the JSON toolkit I am building will make a strong case for it.

The JSONToTable algorithm could also be extended to support NDJSON (newline-delimited JSON), which is common in streaming API responses. FlattenJSON could gain a depth limit parameter. ExtractJSONValue could support JSONPath expressions for more complex queries. All of these are straightforward extensions of the work done in GSoC.

3. Timeline

I understand this is a serious commitment equivalent to a full-time paid internship. I have no other jobs or planned vacations. I do have academic conflicts I want to be transparent about:

- May 1– May 9: End Semester Theory Examinations (100 marks). This is the heaviest exam period. Reduced availability from April 13 to May 9. After May 9 the community bonding period is fully available, no more exams until July.
- July 6: Odd Semester classes resume. From this point I will be attending university alongside coding. I have front-loaded the heavier implementation work in May, June, and early July precisely for this reason.
- August 10 – August 18: Term I Theory Examinations (30 marks). Falls in Weeks 11–12. I have placed integration testing and unit testing in these weeks intentionally ,testing work can be done in shorter focused sessions around exam prep, unlike implementation which needs long uninterrupted blocks.

The timeline is planned with all of these dates in mind. The most demanding implementation work is front-loaded in May, June, and early July when I am free of exams.

Community Bonding Period (May 1 – May 26)

May 1–9: I will still have end-semester exams running, so this window will be lighter. I already have QGIS building locally and will use this time to ensure the development environment is fully configured and continue exploring the relevant parts of the codebase.
May 10–26: Once exams finish, I will be fully available and will focus on deeper codebase exploration, reviewing the relevant Processing and GDAL provider components, and preparing the groundwork for the implementation phase.Read OgrInfo.py, OgrInfoJson.py, and GdalAlgorithm.py in full, to understand how buildArgs() works, how getOgrCompatibleSource() handles different layer types, and how the GDAL version guard pattern works. Study QgsProcessingAlgorithm, QgsProcessingParameterFile, and QgsProcessingOutputString internals. Study QgisAlgorithmProvider.py to understand how new algorithms get registered. Build working local prototypes of LoadJSONFile and ExtractJSONValue and validate them against real ogrinfojson output. Agree final scope, provider placement, and milestone target with Valentin.

Week 1 (May 27 – June 2) Part 1 Setup

Map all usages of EXTRA across OgrInfo.py, OgrInfoJson.py, and dependent tests. Identify exactly which buildArgs() paths need updating for each new parameter. Confirm the GDAL version guard pattern with existing at_least examples in the codebase. Confirm deprecation approach for EXTRA with mentor before writing any code.

Week 2 (June 3 – June 9) Part 1 Implementation

Add typed parameters to both ogrinfo and ogrinfojson: -where (QgsProcessingParameterString), -fid (QgsProcessingParameterNumber integer), -sql (QgsProcessingParameterString), -spat (four QgsProcessingParameterNumber inputs combined in buildArgs()), -geom (QgsProcessingParameterEnum), -fields (QgsProcessingParameterEnum), -limit (QgsProcessingParameterNumber integer). Update buildArgs() to construct GDAL flags from typed inputs, falling back to EXTRA. Handle -sql vs -where mutual exclusion in checkParameterValues().

Week 3 (June 10 – June 16) Part 1 Tests and PR

Mark EXTRA as deprecated with a clear user-facing message. Add the GDAL version guard for -fid (at_least: 3130000). Write YAML test entries for the new parameters in gdal_algorithm_vector_tests.yaml. Open PR for Part 1.

Week 4 (June 17 – June 23) LoadJSONFile

Implement LoadJSONFile: parameter definition in initAlgorithm(), file reading and JSON validation in processAlgorithm(), full error handling for file not found, encoding errors, malformed JSON, and empty file. Register in the algorithm provider. Verify it appears correctly in the model designer UI and its output is connectable as a string input to downstream algorithms.

Week 5 (June 24 – June 30) ExtractJSONValue

Implement ExtractJSONValue: dot-path parser, dict and list index traversal, runtime type detection for string vs number output. Register in provider. Test by connecting ogrinfojson output directly to ExtractJSONValue in a live model to confirm the string-to-extract-to-number chain works end to end. Address any review comments on Part 1 PR.

Week 6 (July 1 – July 7) Integration Testing + Midterm

Integration testing of LoadJSONFile + ExtractJSONValue against real JSON fixtures: ogrinfojson output files, sample API responses, GeoJSON files. Fix edge cases from testing: empty string input, path to missing key, non-UTF-8 files. Midterm evaluation check-in with mentor.

Week 7 (July 8 – July 14) FlattenJSON

Implement FlattenJSON: recursive flattening with dot-separated key paths, array index handling, mixed type handling at each level. Register in provider. Test with real ogrinfojson output to confirm the key paths FlattenJSON produces match what ExtractJSONValue expects.

Week 8 (July 15 – July 21) JSONToTable

Implement JSONToTable: full-array scan for union of keys, QgsFields construction, QgsFeature population with NULL for missing fields, type inference (int to QVariant.Int, float to QVariant.Double, string to QVariant.String, bool to QVariant.Int as 0/1 per QGIS field type constraints, null to NULL). Register in provider. Verify output layer appears in QGIS with correct attribute table.

Week 9 (July 22 – July 28) JSONToTable Stress Testing + Part 2 PR

Stress-test JSONToTable: large arrays (1000+ items), heterogeneous schemas, empty arrays, all-null columns, non-array input (should raise QgsProcessingException with a clear message). Fix type mapping edge cases. Open PR for all Part 2 algorithms.

Week 10 (July 29 – August 4) Demo Models

Build and validate both demo models in the QGIS model designer. Pipeline A: JSON file on disk → LoadJSONFile → ExtractJSONValue → Conditional Branch. Pipeline B: Layer → ogrinfojson → ExtractJSONValue (key: layers.0.featureCount) → Conditional Branch. Save both as .model3 files for inclusion in the repository. Test against real shapefiles with zero and non-zero features.

Week 11 (August 5 – August 11) Integration Tests

Write integration tests for the JSON toolkit algorithms in the QGIS algorithm test suite. Write ogrinfo typed parameter tests in gdal_algorithm_vector_tests.yaml with at_least: 3130000 guard on -fid. Run the full test suite locally to confirm no regressions. I have Term I university exams running August 10–18 but they are light 30-mark tests and I am comfortable working alongside them.

Week 12 (August 12 – August 18) Unit Tests

Write pytest unit tests for the JSON parsing logic independent of QGIS, covering: LoadJSONFile (valid UTF-8 JSON, malformed JSON, file not found, empty file), ExtractJSONValue (nested dict traversal, array index access, missing key, numeric vs string detection), FlattenJSON (nested dict, array-of-objects, empty object), JSONToTable (feature count, field derivation, type inference, empty array, mixed-schema arrays).

Week 13 (August 19 – August 25) Documentation

Write user-facing documentation for all four JSON algorithms following the QGIS algorithm documentation format. Document the new typed ogrinfo parameters. Write a worked example narrative for both demo models explaining the end-to-end flow.

Week 14 (August 26 – September 1) Final

Address any review comments across all PRs. Final polish to ensure all tests pass on CI. Submit the GSoC final evaluation.

4. Studies

School and Degree

B.Tech in Computer Science and Engineering (IoT Specialisation),
Institute of Engineering and Management (IEM), Salt Lake, Kolkata, India.
Currently in third year (2023–2027). CGPA: **9.2/10.**

Higher Secondary | Delhi Public School Ruby Park, Kolkata | CBSE (2023) | Score: **94%**

Secondary (Class X) | Delhi Public School Megacity, Kolkata | ICSE (2021) | Score: **95%**

Contribution to Studies

My IoT specialisation focuses on data pipelines, sensor streams, and typed data flow. This is exactly what the processing framework is about at a conceptual level. Working on the QGIS processing

framework directly extends what I am studying: how typed data moves through a system, how you design clean APIs for data transformation, and how you test pipeline components in isolation. The experience of working in a large production Python codebase with real code review is also something my coursework does not give me.

5. Programming and GIS Experience

Computing Experience

I work on Windows 11 daily and use Ubuntu via WSL for open source work. Languages I use regularly: Python (strongest), C++ (enough to read and navigate a C++ codebasebut I would not call myself fluent), Java, TypeScript, JavaScript. I have built full-stack applications with Node.js, Express, React, Next.js, and Flask. I am comfortable with Git PR workflows, CMake, and conda environments. For the internship at iQuester I worked with InfluxDB for time-series data and built REST APIs for IoT sensor ingestion.

GIS Experience as a User

My exposure to GIS has primarily come through contributing to the QGIS and GDAL codebases and studying how geospatial data flows through these systems. Through this work I have explored QGIS, especially the Processing framework and Model Designer, to understand how algorithms connect and how vector layer inputs move through processing workflows.

Before contributing to QGIS, my interaction with geospatial data was mainly through web development projects where I worked with GeoJSON and read about common geospatial formats and tools. Contributing to the QGIS ecosystem has since given me deeper familiarity with how GDAL-backed algorithms integrate with the Processing framework and how geospatial data is handled within the application.

GIS Programming and Open Source Contributions

I have been contributing to GDAL since early 2026. My contributions include implementing new CLI features in GDALVectorInfo, improving the test infrastructure by marking network-dependent tests correctly to prevent false CI failures, and adding documentation improvements such as reference tables and build fixes. One of my patches was also backported to the stable release branch.

The most substantial contribution was adding a new command-line flag to **gdal vector info**, which was developed in collaboration with the GDAL core maintainer Even Rouault. During the review processes he provided his valuable guidance on community norms, testing practices, and how to structure changes for maintainability. This experience helped me understand how large open-source projects review contributions and maintain production-quality C++ code.

Full list of merged contributions:
https://github.com/OSGeo/gdal/pulls?q=author%3ASionigdha+is%3Amerged

On the QGIS side, I have contributed documentation improvements targeting the QGIS 4.0 development cycle. These contributions include clarifying the initialization behavior of the `QgsLogger` class and improving related documentation to make the behavior clearer for developers.

List of merged QGIS contributions:
https://github.com/qgis/QGIS/pulls?q=author%3ASionigdha+is%3Amerged

6. GSoC Participation

Have you participated in GSoC before? No, this is my first time applying.

Have you applied but were not selected? No.

Will you submit another proposal to a different org this year? No. This is my only GSoC application for 2026.