

## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
  - Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
    - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
    - Ensure compatibility with hi2temp and make necessary adjustments.
  - **Week 6: (July 7 – July 13)**
    - Conduct comprehensive testing on all newly implemented functionalities.
    - Verify that nothing breaks in the existing codebase.
  - **Week 7: (July 14 – July 20)**
    - Focus on extensive testing to improve code coverage and ensure stability.
    - Address edge cases and unexpected scenarios.
  - **Week 8: (July 21 – July 27)**
    - Identify any remaining bottlenecks in the pipeline.
    - Optimize slow components using Numba or CuPy wherever necessary.
  - **Week 9: (July 28 – Aug 5)**
    - Add detailed docstrings and update documentation for all changes.
    - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

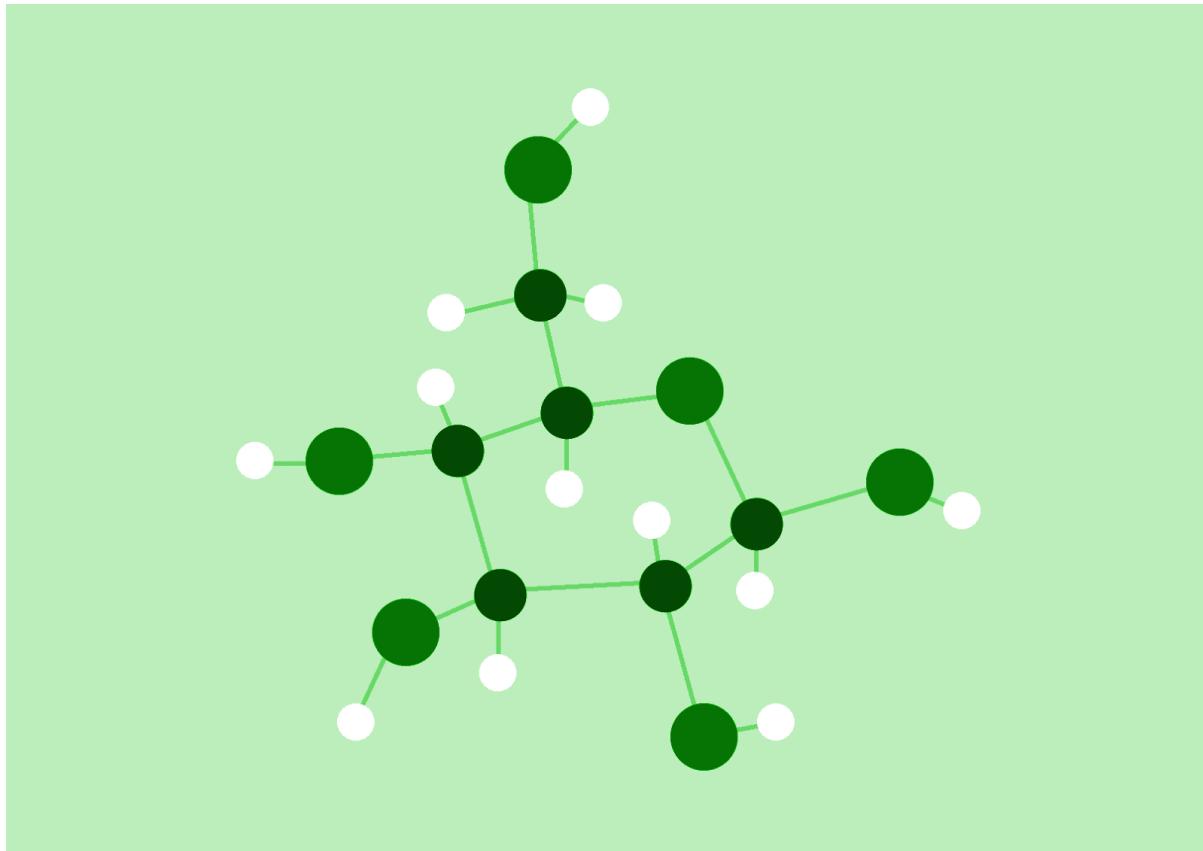
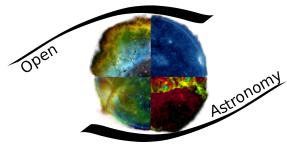
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
  - Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
    - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
    - Ensure compatibility with hi2temp and make necessary adjustments.
  - **Week 6: (July 7 – July 13)**
    - Conduct comprehensive testing on all newly implemented functionalities.
    - Verify that nothing breaks in the existing codebase.
  - **Week 7: (July 14 – July 20)**
    - Focus on extensive testing to improve code coverage and ensure stability.
    - Address edge cases and unexpected scenarios.
  - **Week 8: (July 21 – July 27)**
    - Identify any remaining bottlenecks in the pipeline.
    - Optimize slow components using Numba or CuPy wherever necessary.
  - **Week 9: (July 28 – Aug 5)**
    - Add detailed docstrings and update documentation for all changes.
    - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

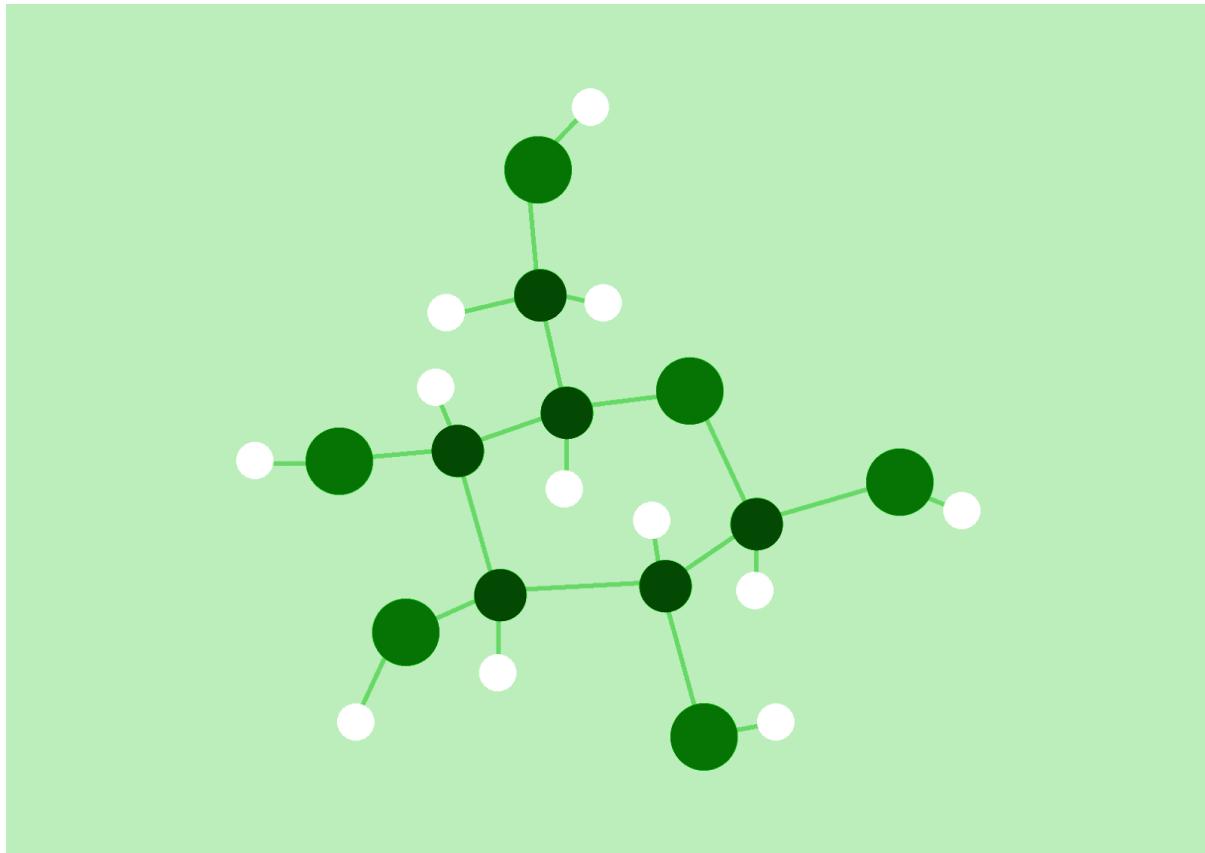
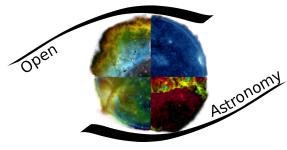
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

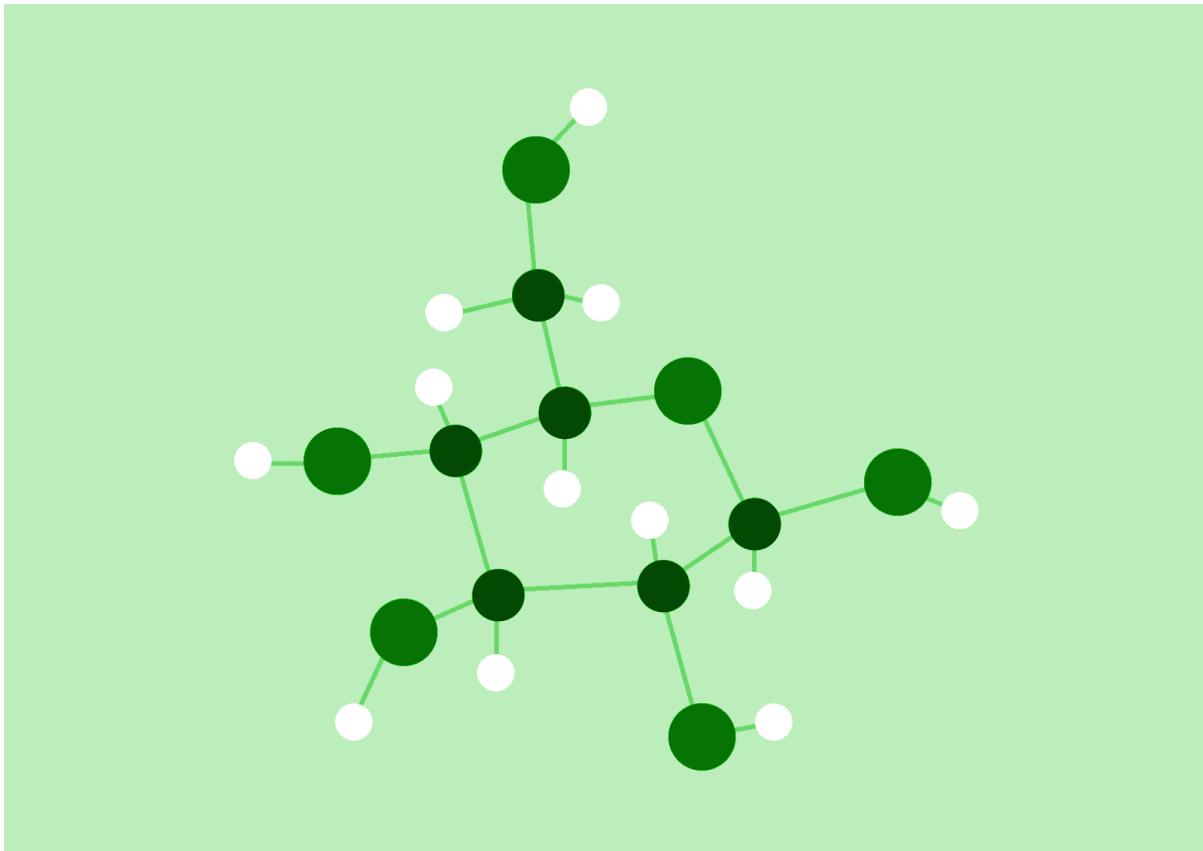
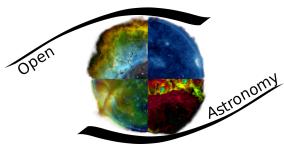
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
  - Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
    - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
    - Ensure compatibility with hi2temp and make necessary adjustments.
  - **Week 6: (July 7 – July 13)**
    - Conduct comprehensive testing on all newly implemented functionalities.
    - Verify that nothing breaks in the existing codebase.
  - **Week 7: (July 14 – July 20)**
    - Focus on extensive testing to improve code coverage and ensure stability.
    - Address edge cases and unexpected scenarios.
  - **Week 8: (July 21 – July 27)**
    - Identify any remaining bottlenecks in the pipeline.
    - Optimize slow components using Numba or CuPy wherever necessary.
  - **Week 9: (July 28 – Aug 5)**
    - Add detailed docstrings and update documentation for all changes.
    - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

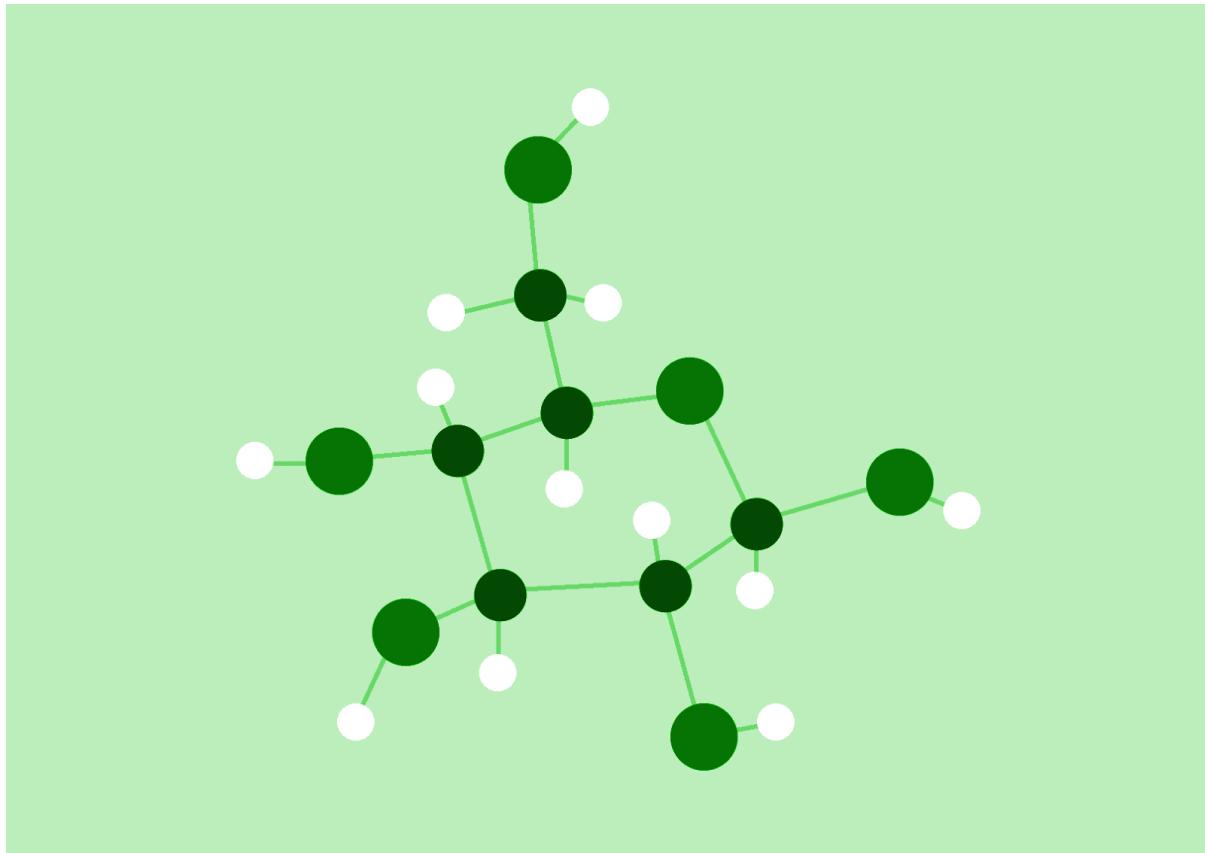
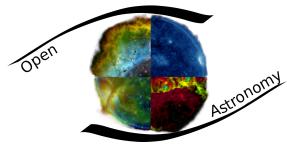
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

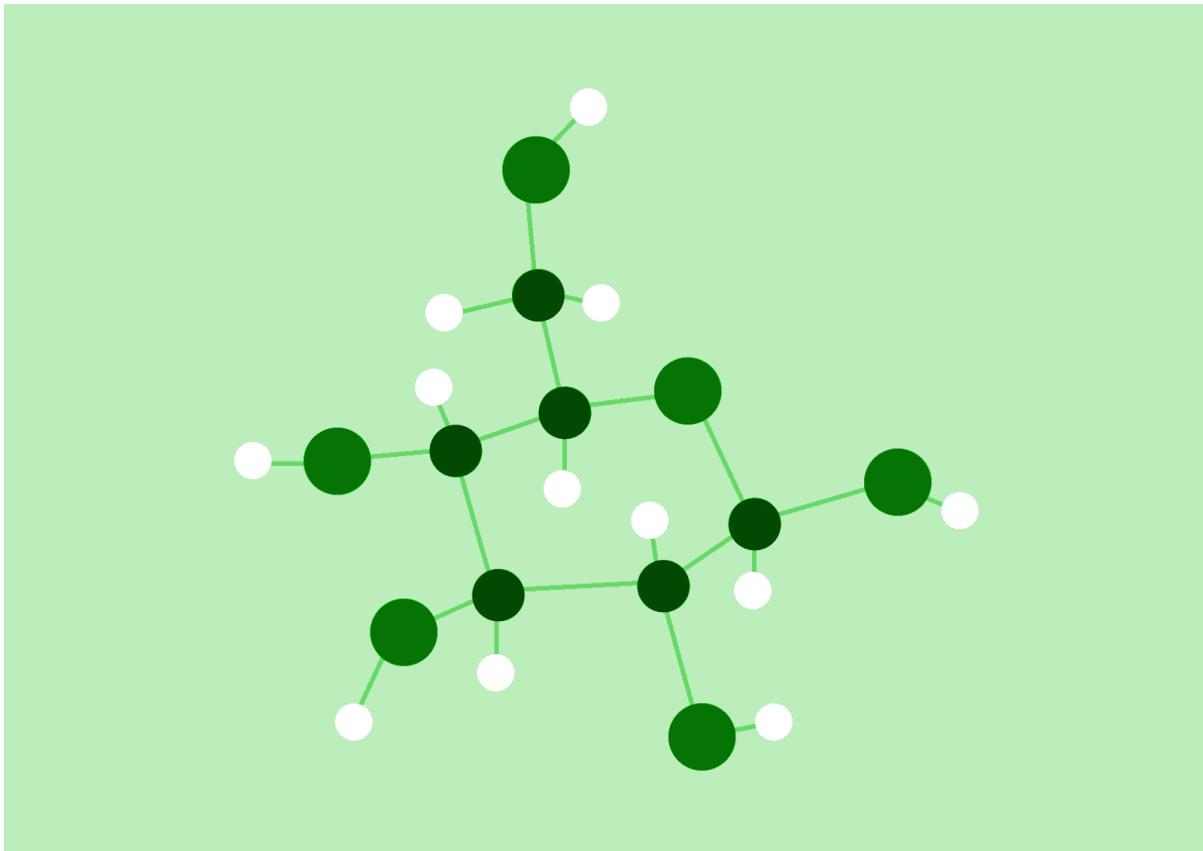
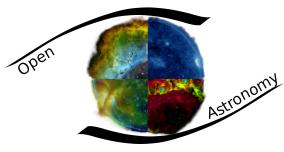
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
  - Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
    - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
    - Ensure compatibility with hi2temp and make necessary adjustments.
  - **Week 6: (July 7 – July 13)**
    - Conduct comprehensive testing on all newly implemented functionalities.
    - Verify that nothing breaks in the existing codebase.
  - **Week 7: (July 14 – July 20)**
    - Focus on extensive testing to improve code coverage and ensure stability.
    - Address edge cases and unexpected scenarios.
  - **Week 8: (July 21 – July 27)**
    - Identify any remaining bottlenecks in the pipeline.
    - Optimize slow components using Numba or CuPy wherever necessary.
  - **Week 9: (July 28 – Aug 5)**
    - Add detailed docstrings and update documentation for all changes.
    - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

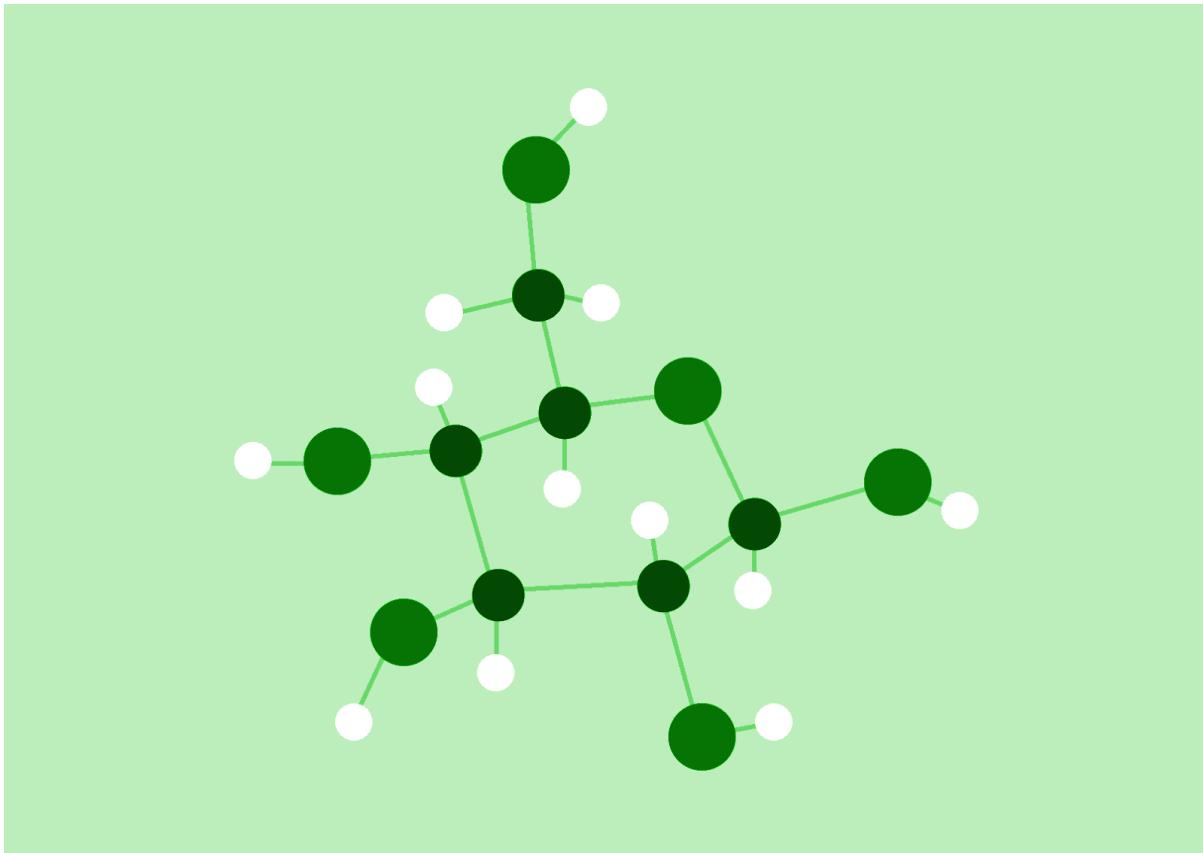
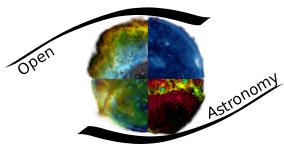
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
  - Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
    - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
    - Ensure compatibility with hi2temp and make necessary adjustments.
  - **Week 6: (July 7 – July 13)**
    - Conduct comprehensive testing on all newly implemented functionalities.
    - Verify that nothing breaks in the existing codebase.
  - **Week 7: (July 14 – July 20)**
    - Focus on extensive testing to improve code coverage and ensure stability.
    - Address edge cases and unexpected scenarios.
  - **Week 8: (July 21 – July 27)**
    - Identify any remaining bottlenecks in the pipeline.
    - Optimize slow components using Numba or CuPy wherever necessary.
  - **Week 9: (July 28 – Aug 5)**
    - Add detailed docstrings and update documentation for all changes.
    - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

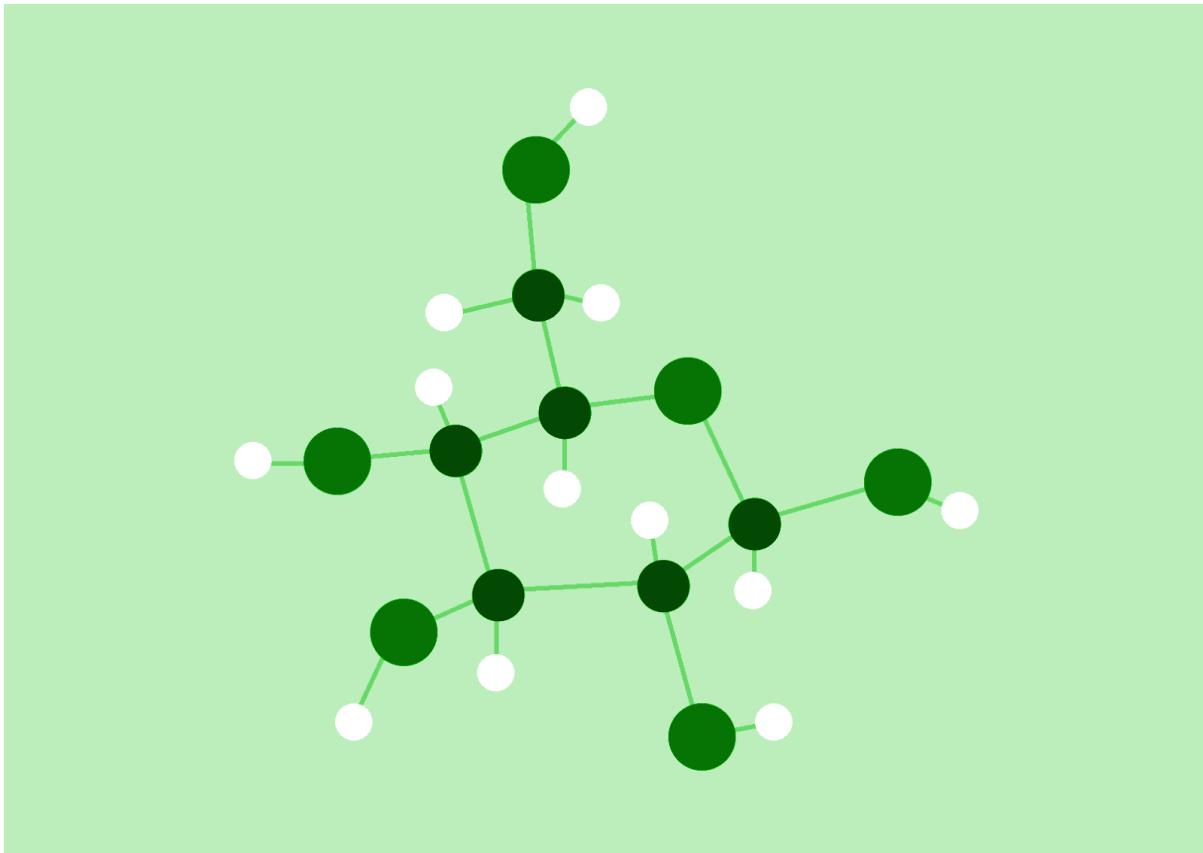
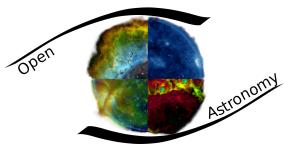
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
  - Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
    - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
    - Ensure compatibility with hi2temp and make necessary adjustments.
  - **Week 6: (July 7 – July 13)**
    - Conduct comprehensive testing on all newly implemented functionalities.
    - Verify that nothing breaks in the existing codebase.
  - **Week 7: (July 14 – July 20)**
    - Focus on extensive testing to improve code coverage and ensure stability.
    - Address edge cases and unexpected scenarios.
  - **Week 8: (July 21 – July 27)**
    - Identify any remaining bottlenecks in the pipeline.
    - Optimize slow components using Numba or CuPy wherever necessary.
  - **Week 9: (July 28 – Aug 5)**
    - Add detailed docstrings and update documentation for all changes.
    - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

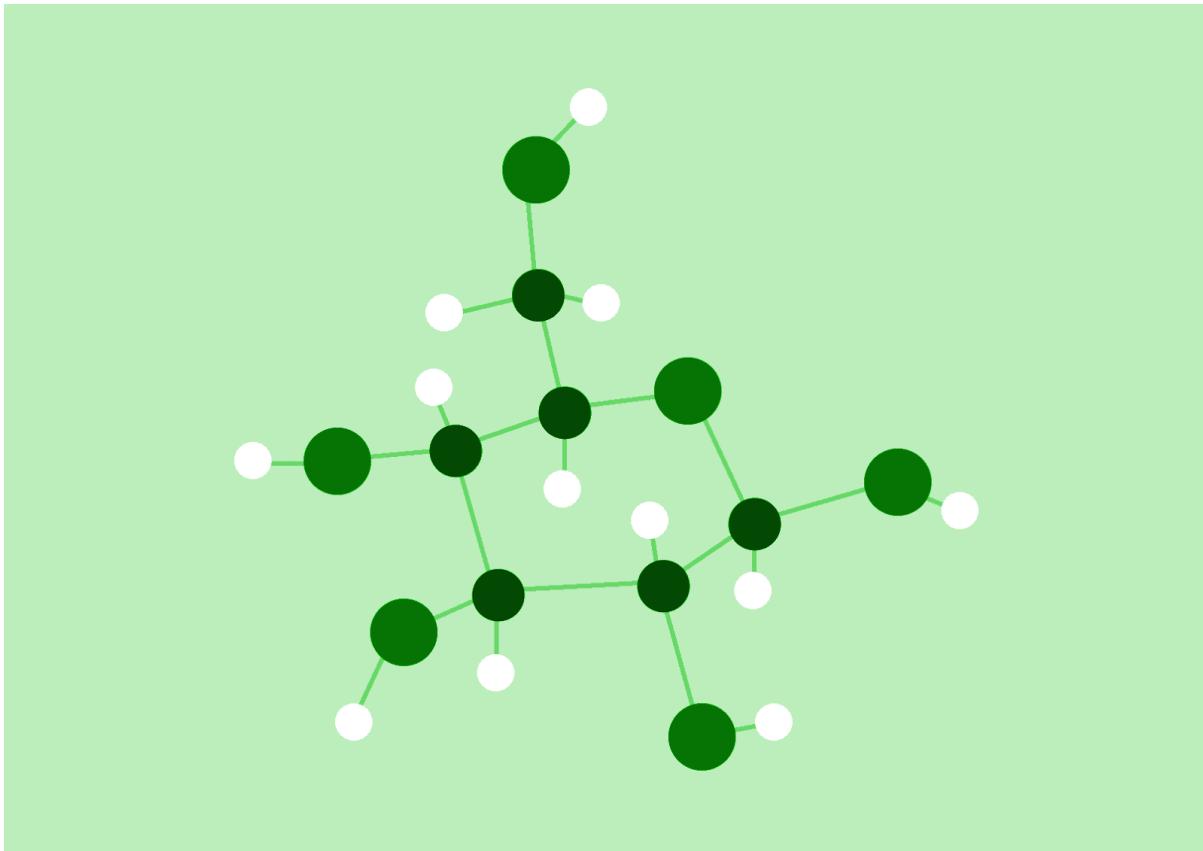
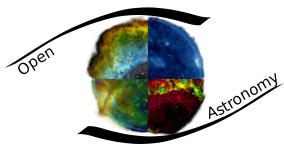
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

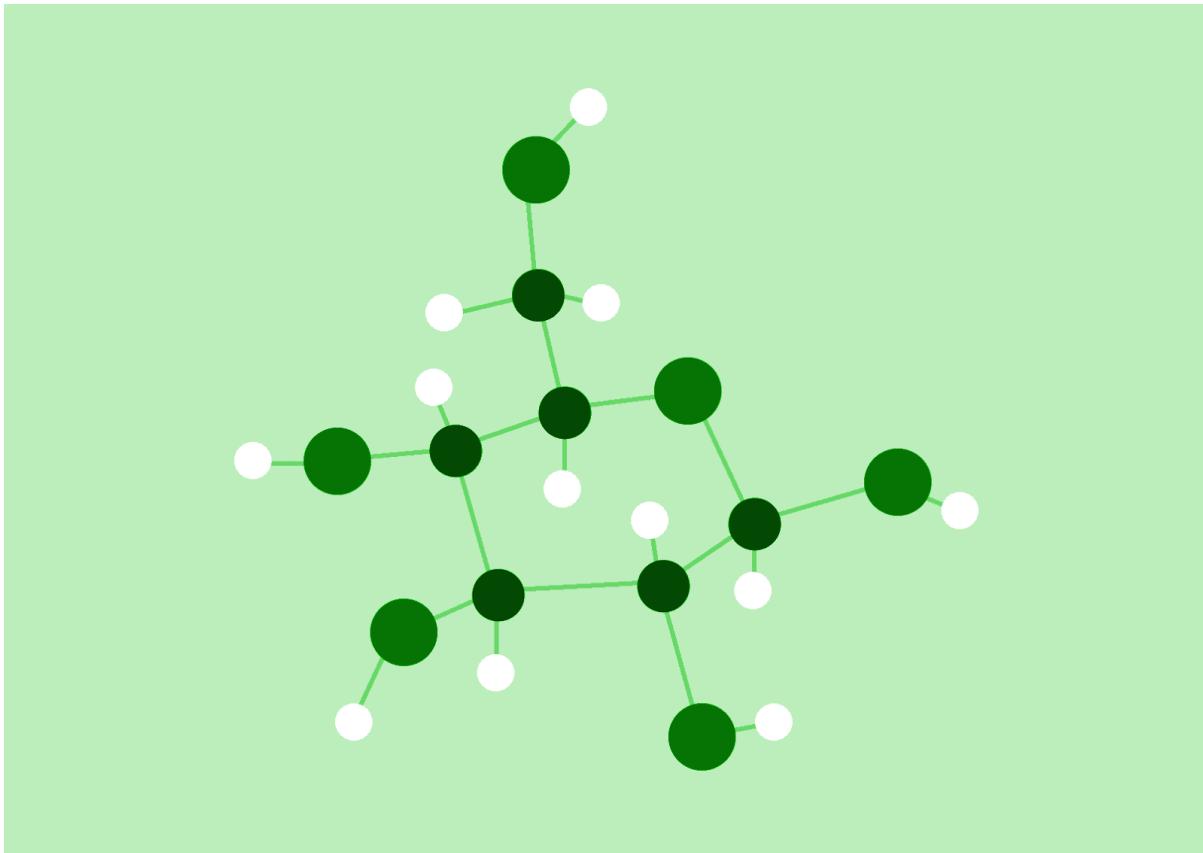
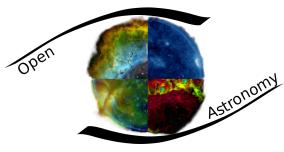
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

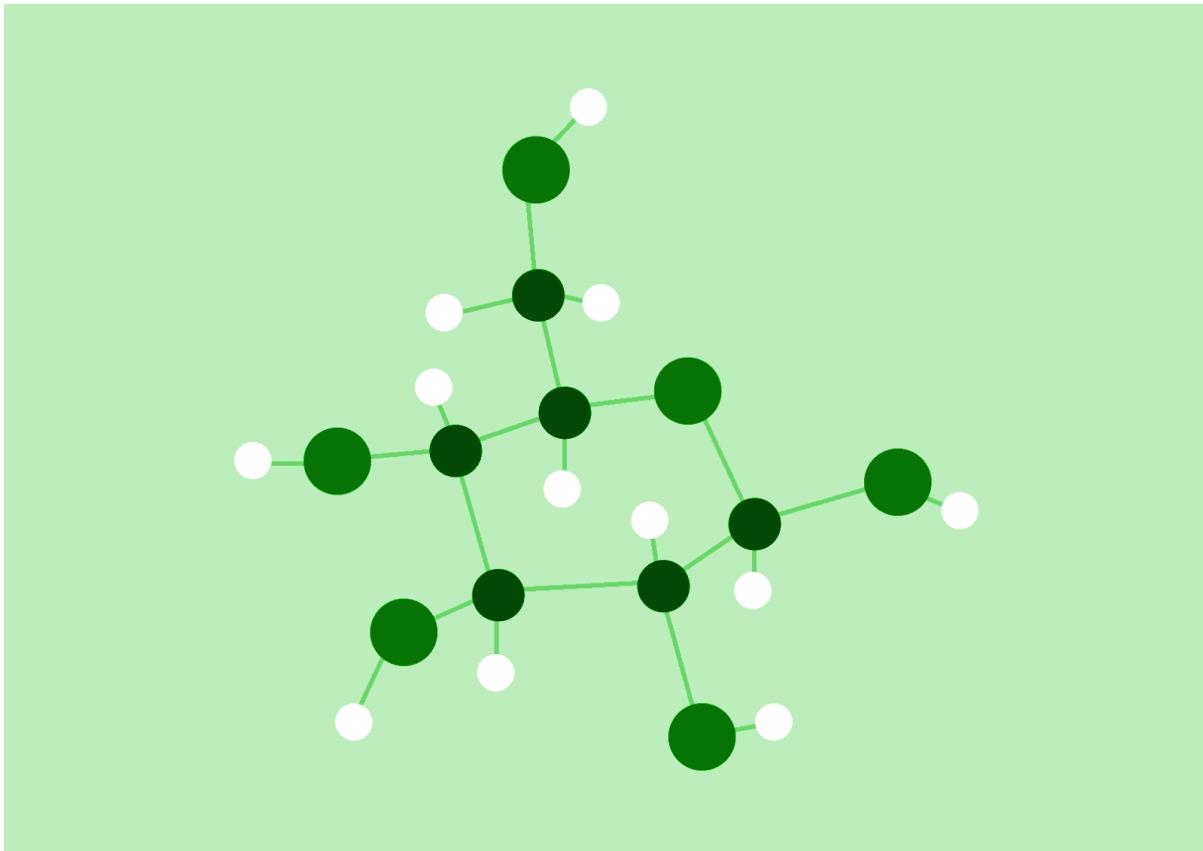
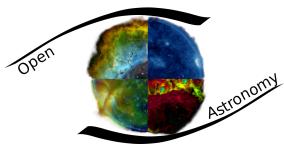
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
  - Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
    - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
    - Ensure compatibility with hi2temp and make necessary adjustments.
  - **Week 6: (July 7 – July 13)**
    - Conduct comprehensive testing on all newly implemented functionalities.
    - Verify that nothing breaks in the existing codebase.
  - **Week 7: (July 14 – July 20)**
    - Focus on extensive testing to improve code coverage and ensure stability.
    - Address edge cases and unexpected scenarios.
  - **Week 8: (July 21 – July 27)**
    - Identify any remaining bottlenecks in the pipeline.
    - Optimize slow components using Numba or CuPy wherever necessary.
  - **Week 9: (July 28 – Aug 5)**
    - Add detailed docstrings and update documentation for all changes.
    - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

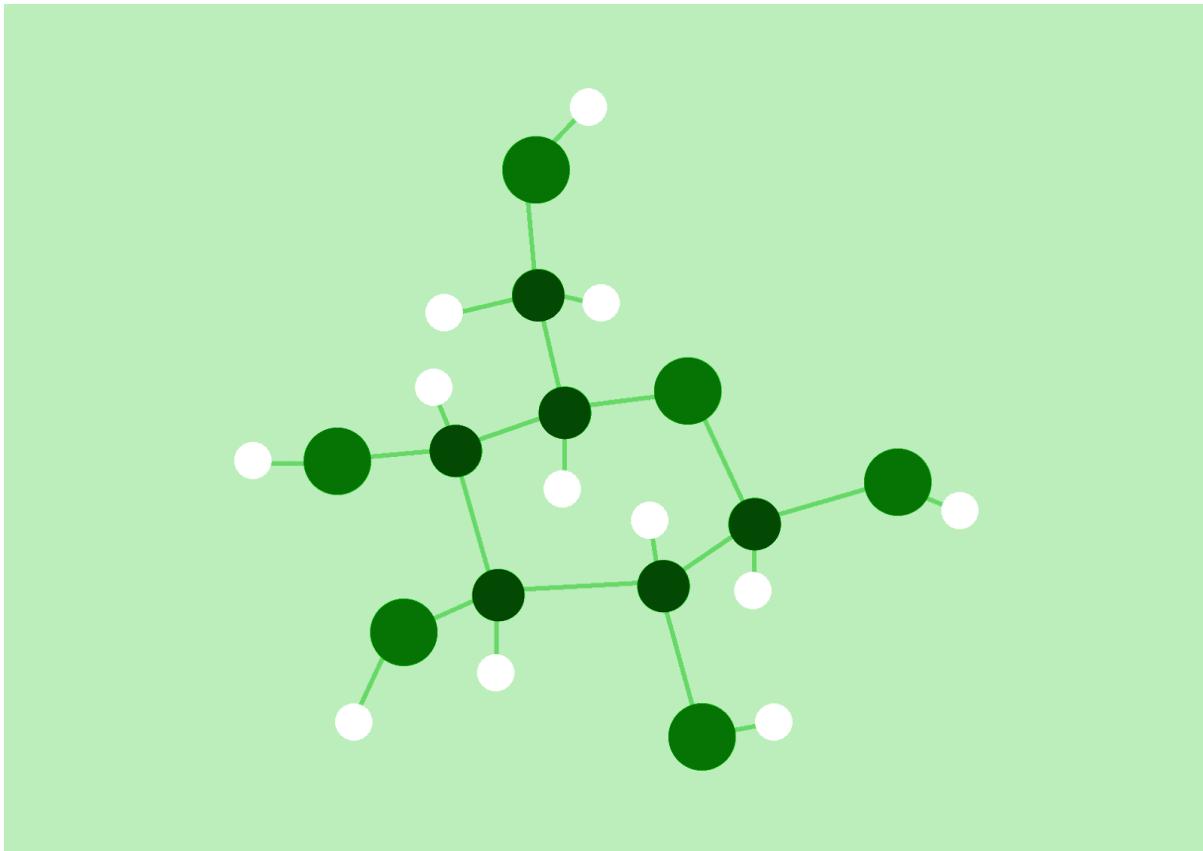
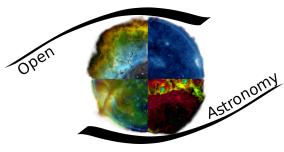
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

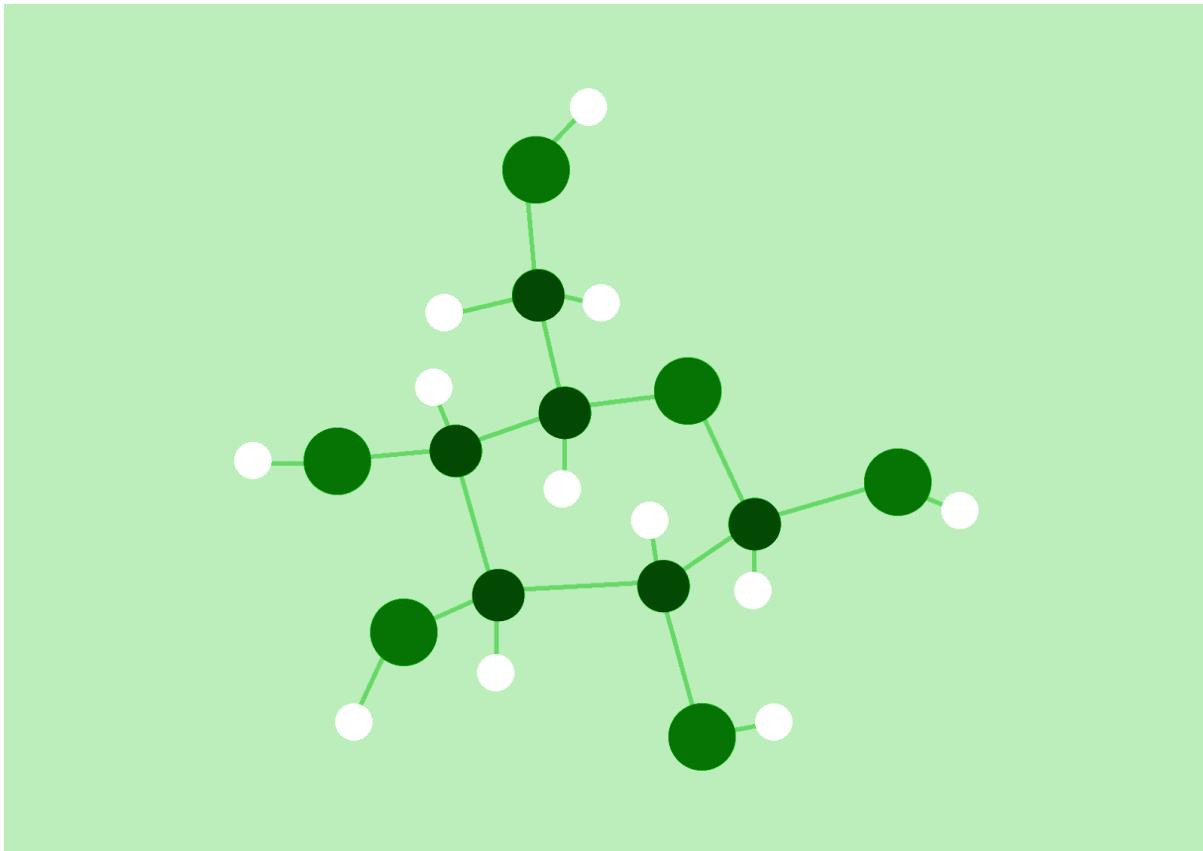
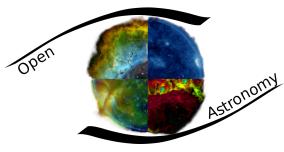
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



Radis

Fast parsing of large databases and  
execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
  - Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
    - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
    - Ensure compatibility with hi2temp and make necessary adjustments.
  - **Week 6: (July 7 – July 13)**
    - Conduct comprehensive testing on all newly implemented functionalities.
    - Verify that nothing breaks in the existing codebase.
  - **Week 7: (July 14 – July 20)**
    - Focus on extensive testing to improve code coverage and ensure stability.
    - Address edge cases and unexpected scenarios.
  - **Week 8: (July 21 – July 27)**
    - Identify any remaining bottlenecks in the pipeline.
    - Optimize slow components using Numba or CuPy wherever necessary.
  - **Week 9: (July 28 – Aug 5)**
    - Add detailed docstrings and update documentation for all changes.
    - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

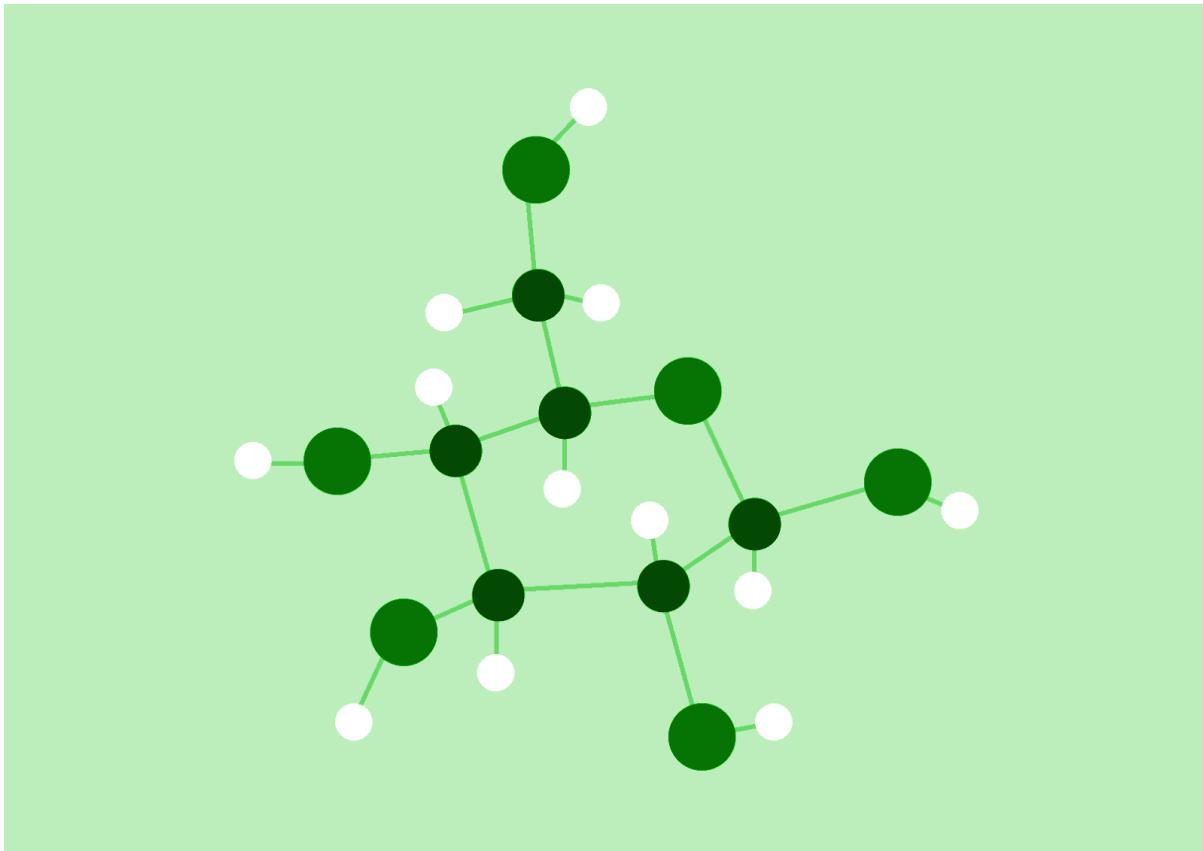
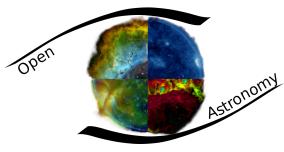
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



Radis

Fast parsing of large databases and  
execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

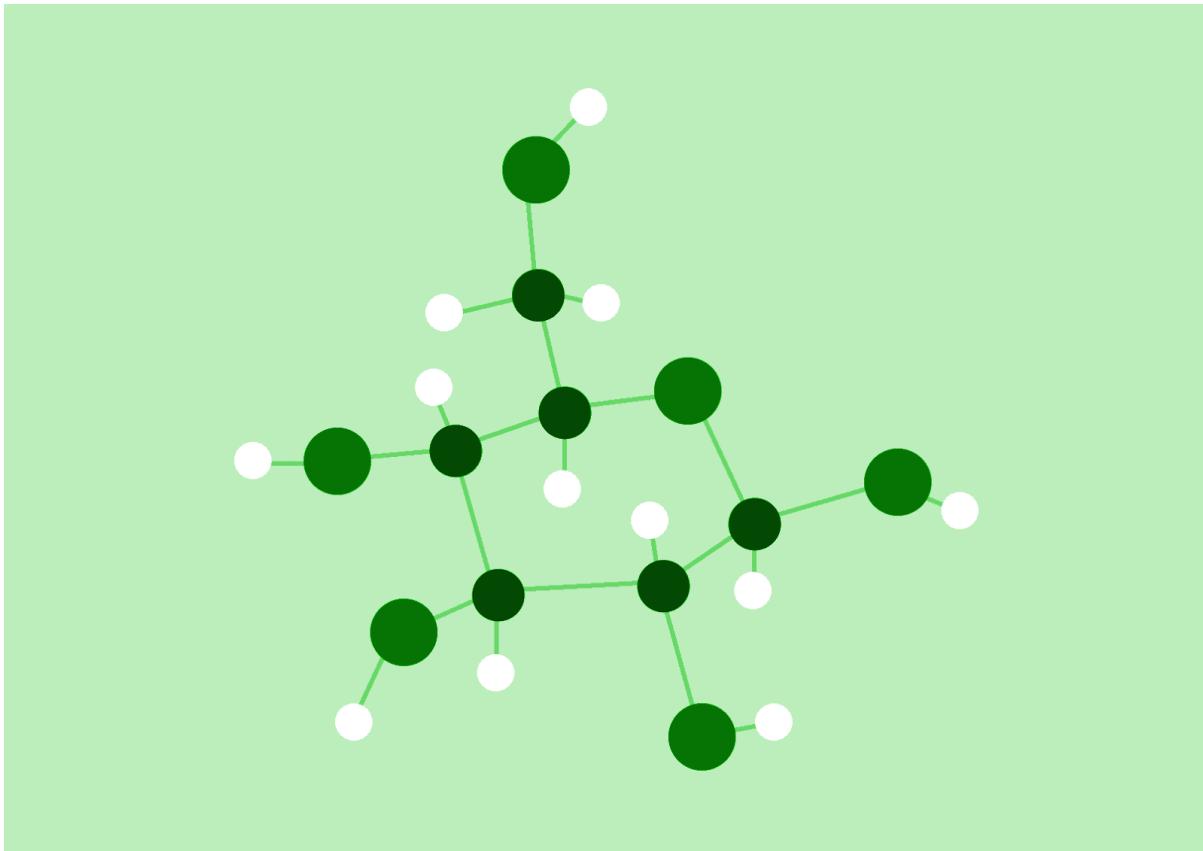
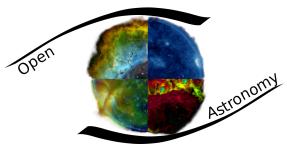
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



Radis

Fast parsing of large databases and  
execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

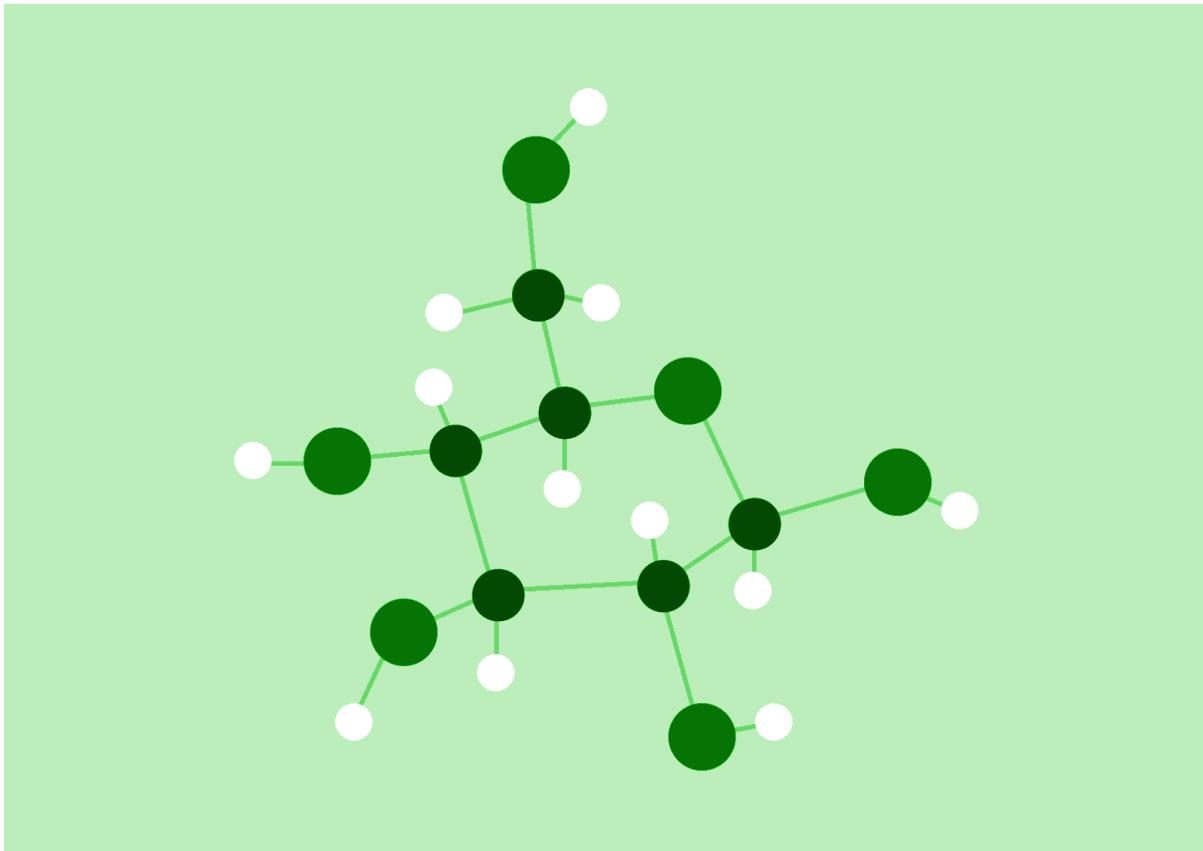
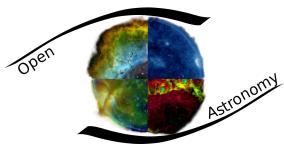
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

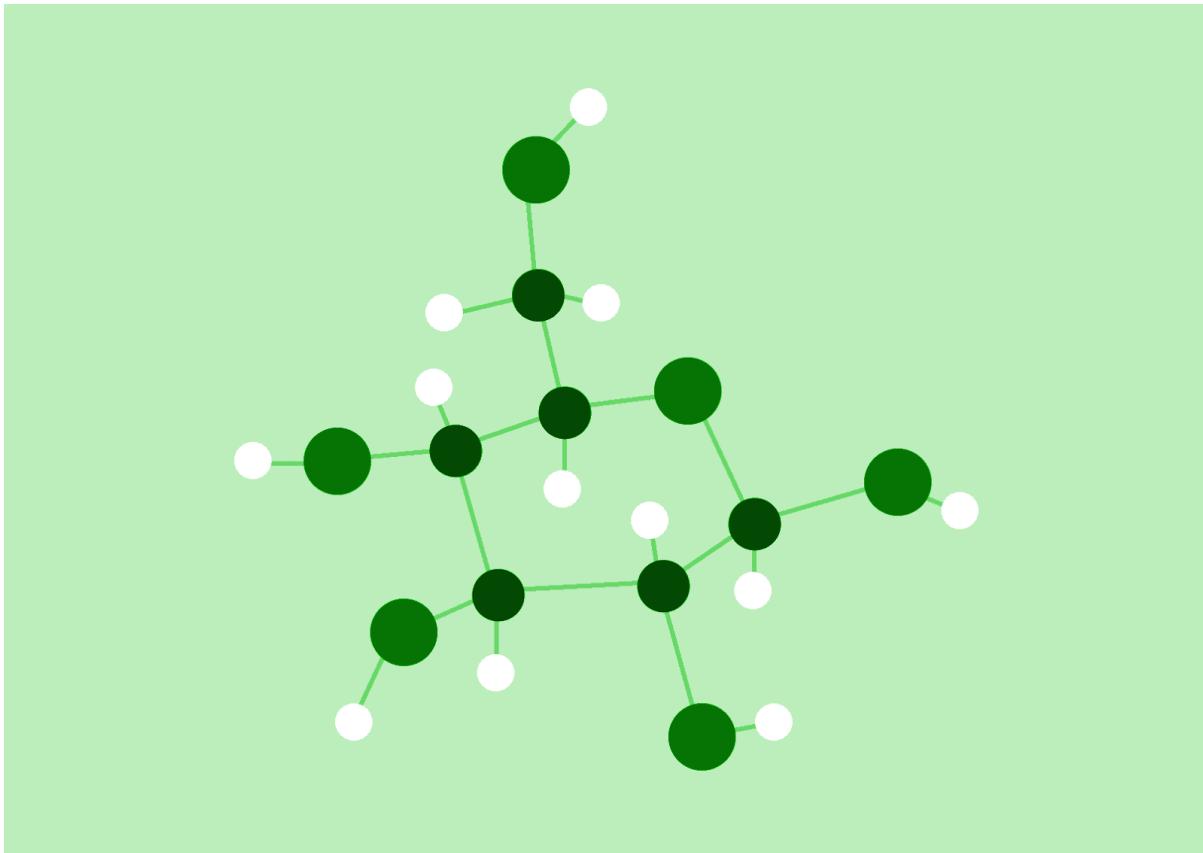
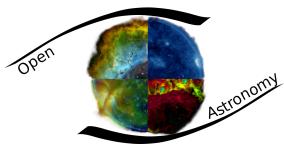
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

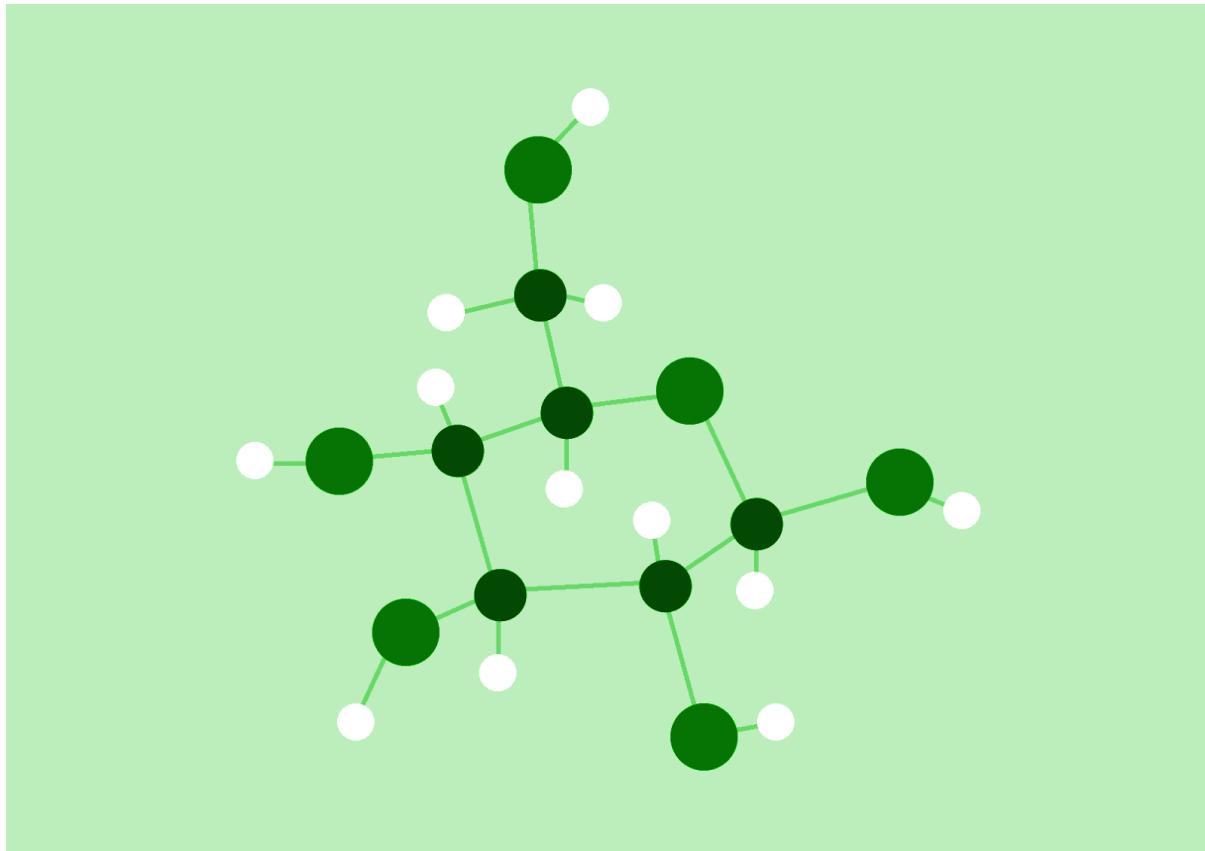
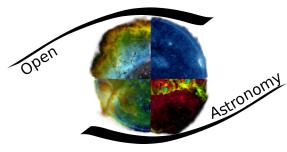
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

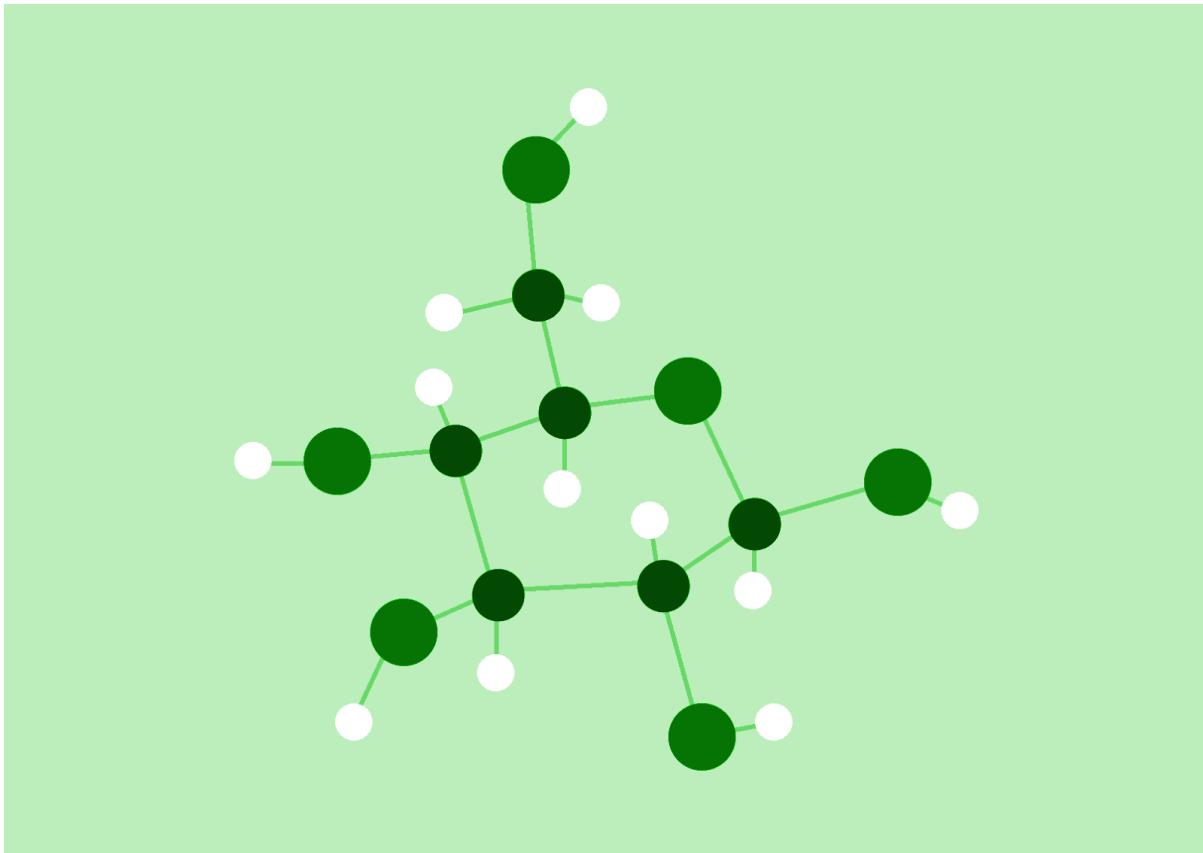
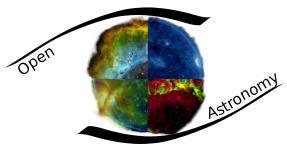
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

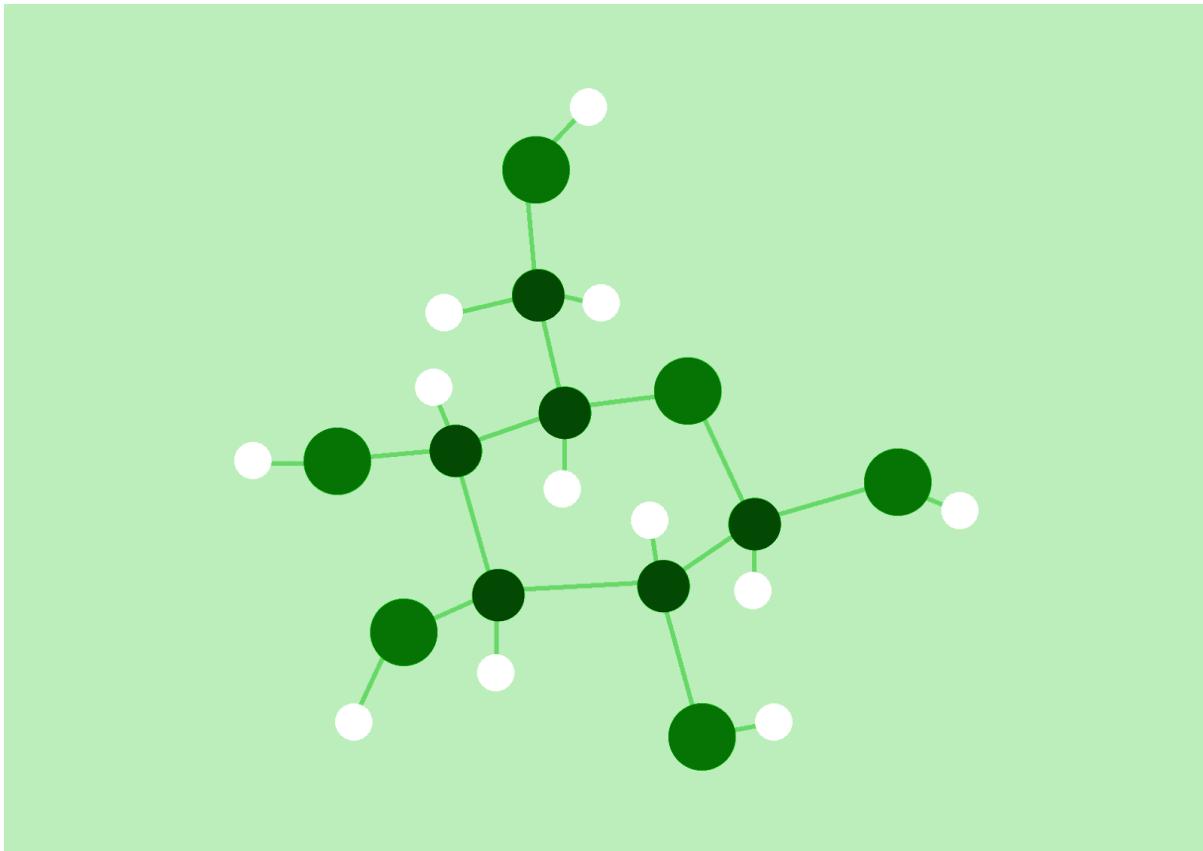
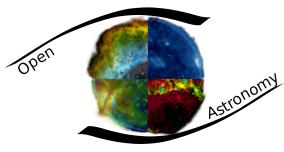
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
  - Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
    - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
    - Ensure compatibility with hi2temp and make necessary adjustments.
  - **Week 6: (July 7 – July 13)**
    - Conduct comprehensive testing on all newly implemented functionalities.
    - Verify that nothing breaks in the existing codebase.
  - **Week 7: (July 14 – July 20)**
    - Focus on extensive testing to improve code coverage and ensure stability.
    - Address edge cases and unexpected scenarios.
  - **Week 8: (July 21 – July 27)**
    - Identify any remaining bottlenecks in the pipeline.
    - Optimize slow components using Numba or CuPy wherever necessary.
  - **Week 9: (July 28 – Aug 5)**
    - Add detailed docstrings and update documentation for all changes.
    - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

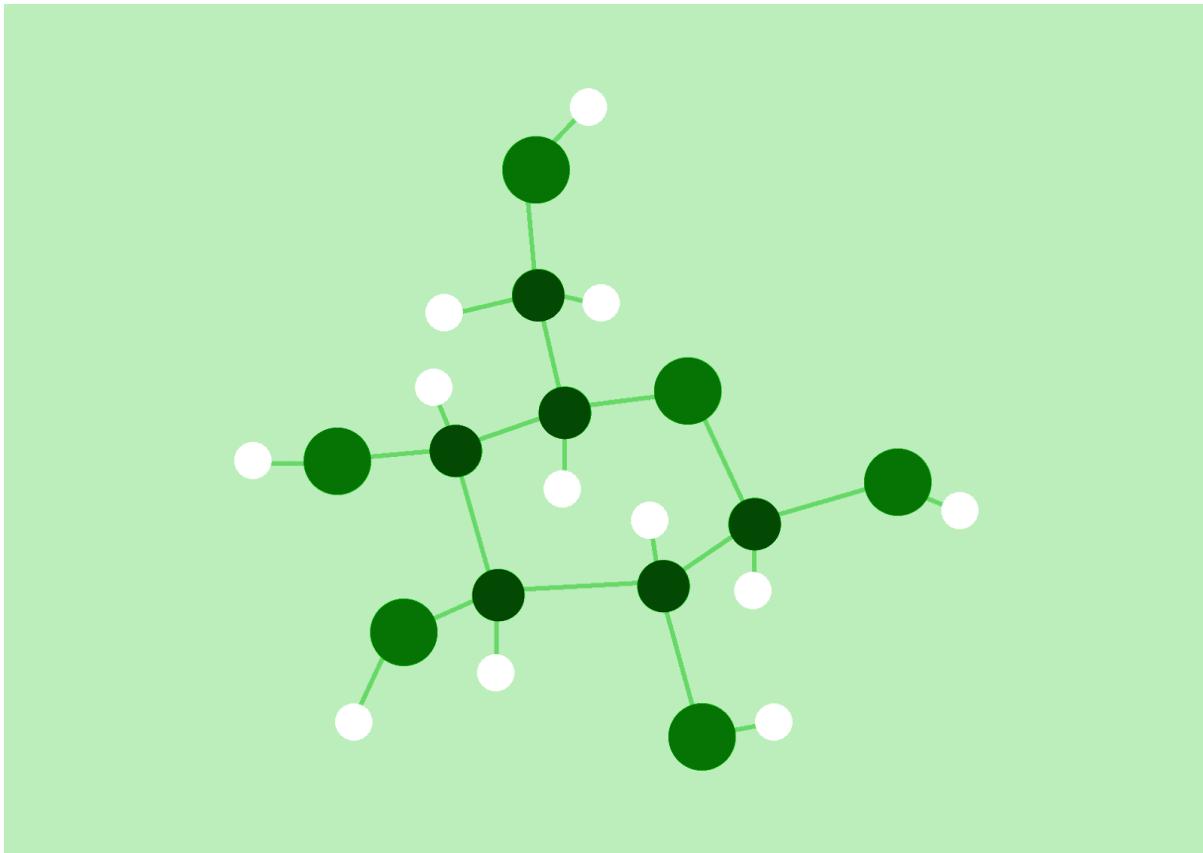
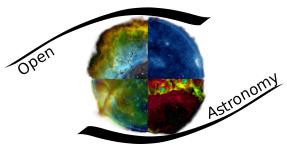
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



Radis

Fast parsing of large databases and  
execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

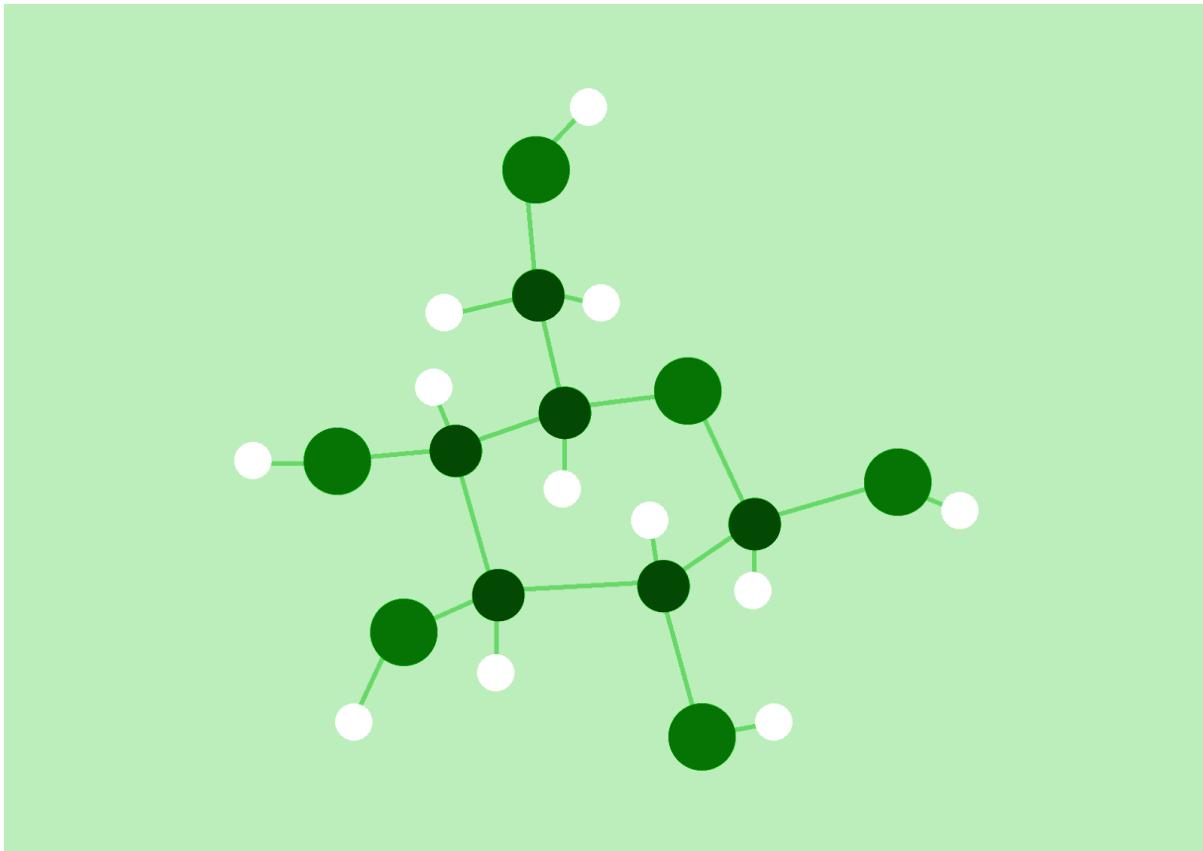
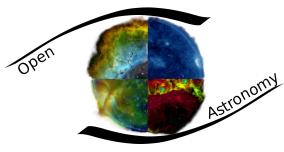
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

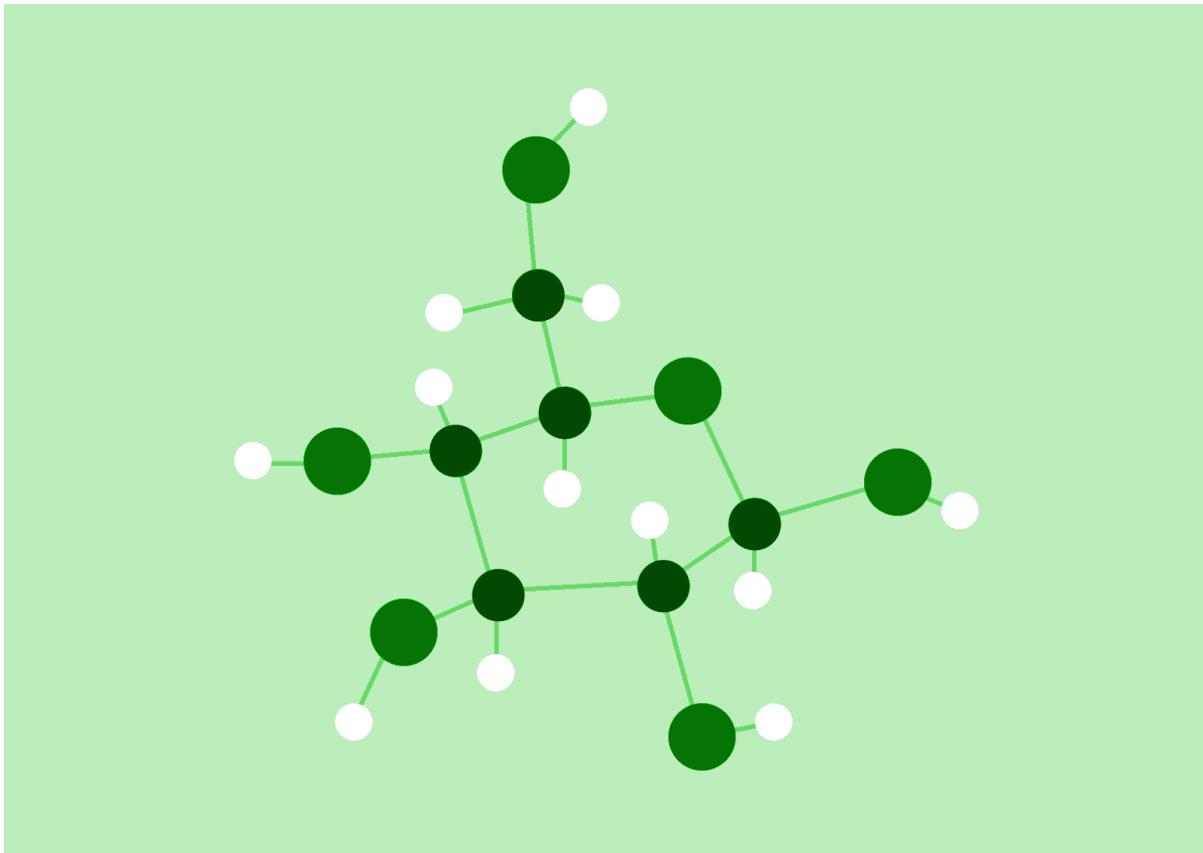
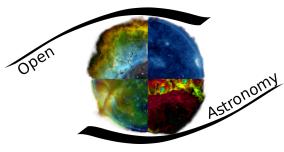
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

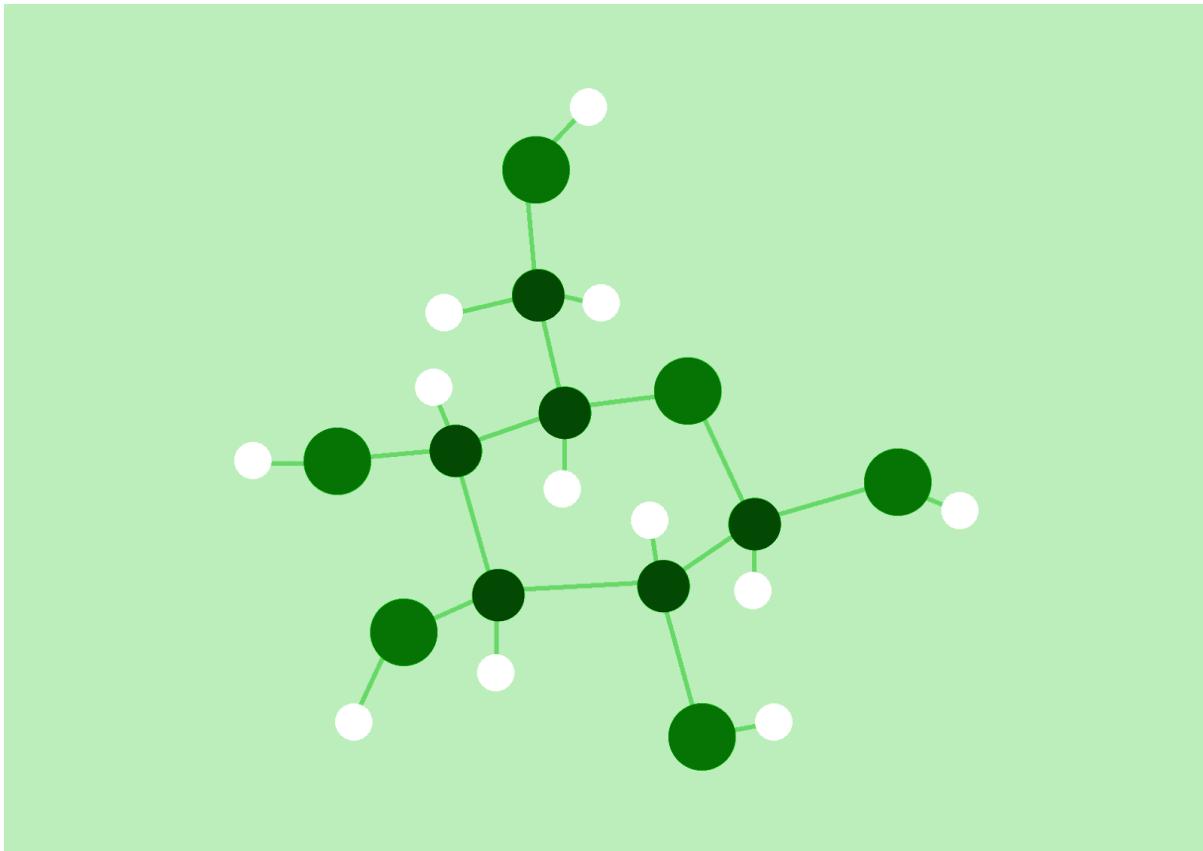
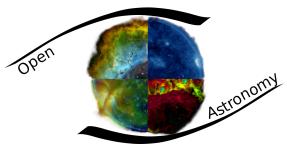
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



Radis

Fast parsing of large databases and  
execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

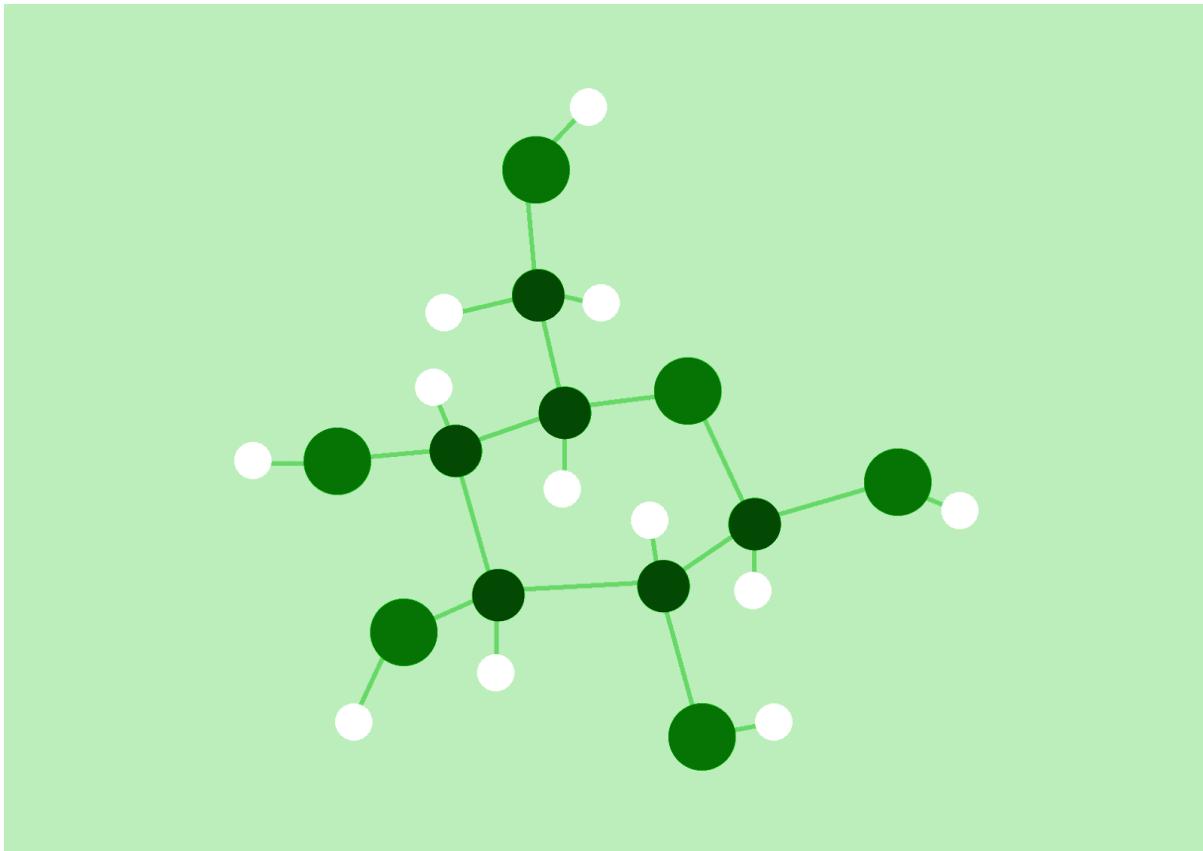
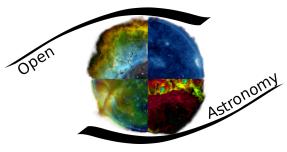
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



Radis

Fast parsing of large databases and  
execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
  - Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
    - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
    - Ensure compatibility with hi2temp and make necessary adjustments.
  - **Week 6: (July 7 – July 13)**
    - Conduct comprehensive testing on all newly implemented functionalities.
    - Verify that nothing breaks in the existing codebase.
  - **Week 7: (July 14 – July 20)**
    - Focus on extensive testing to improve code coverage and ensure stability.
    - Address edge cases and unexpected scenarios.
  - **Week 8: (July 21 – July 27)**
    - Identify any remaining bottlenecks in the pipeline.
    - Optimize slow components using Numba or CuPy wherever necessary.
  - **Week 9: (July 28 – Aug 5)**
    - Add detailed docstrings and update documentation for all changes.
    - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

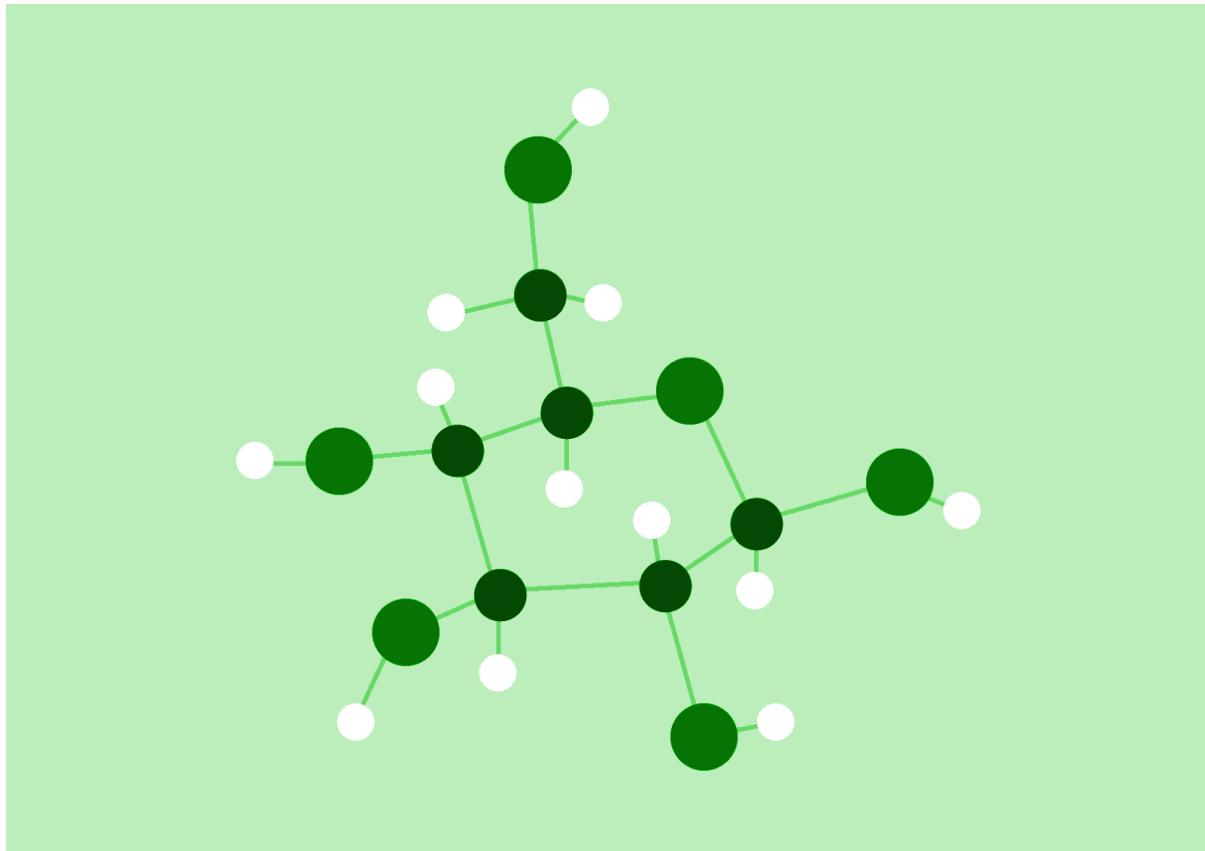
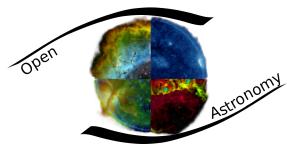
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

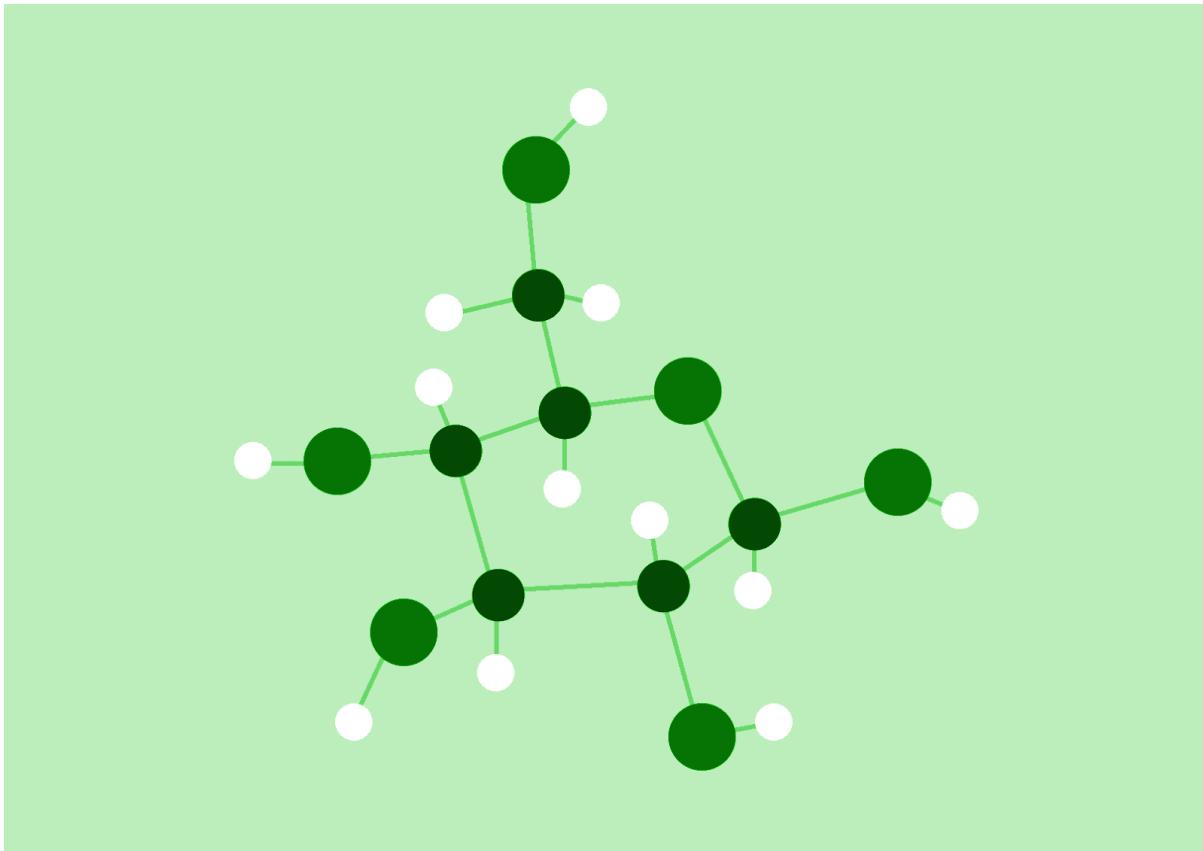
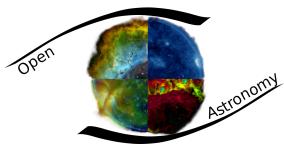
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



Radis

Fast parsing of large databases and  
execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
  - Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
    - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
    - Ensure compatibility with hi2temp and make necessary adjustments.
  - **Week 6: (July 7 – July 13)**
    - Conduct comprehensive testing on all newly implemented functionalities.
    - Verify that nothing breaks in the existing codebase.
  - **Week 7: (July 14 – July 20)**
    - Focus on extensive testing to improve code coverage and ensure stability.
    - Address edge cases and unexpected scenarios.
  - **Week 8: (July 21 – July 27)**
    - Identify any remaining bottlenecks in the pipeline.
    - Optimize slow components using Numba or CuPy wherever necessary.
  - **Week 9: (July 28 – Aug 5)**
    - Add detailed docstrings and update documentation for all changes.
    - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

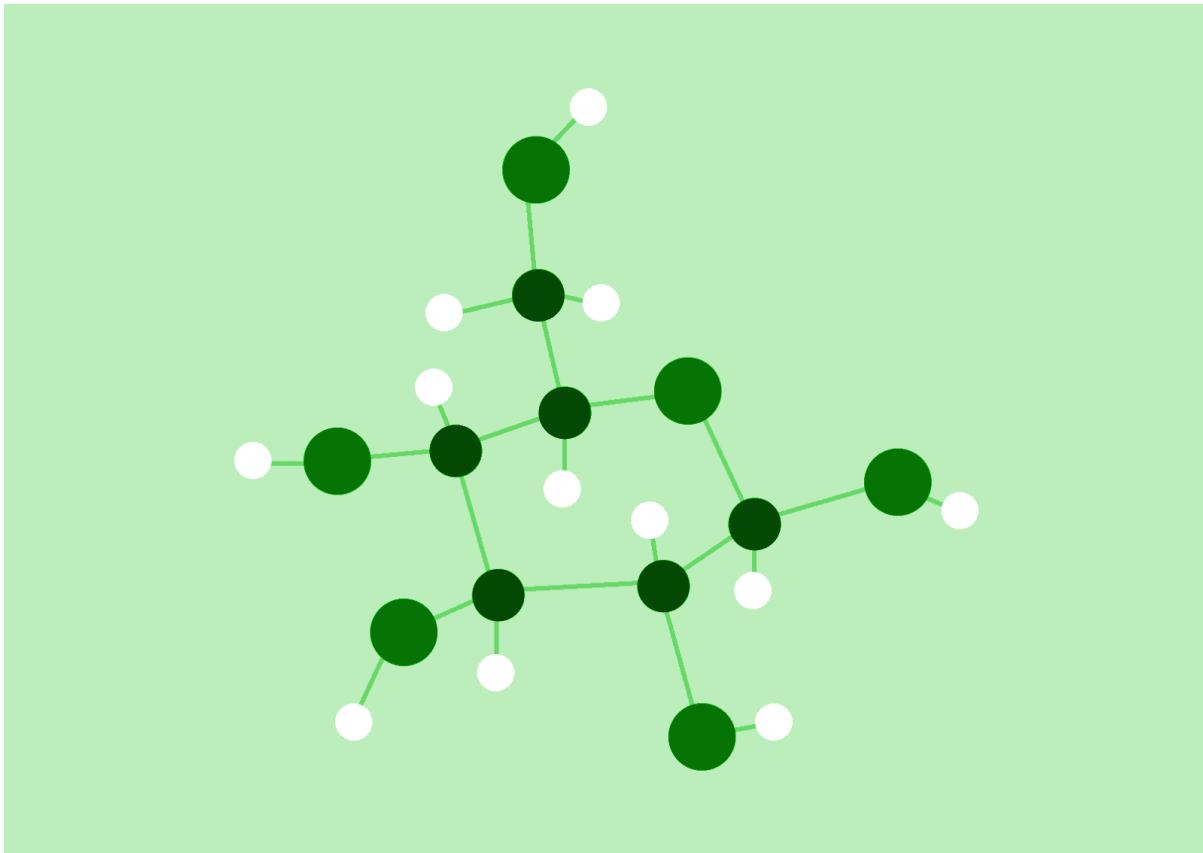
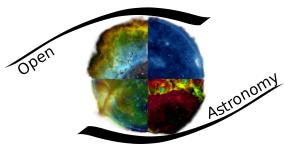
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
  - Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
    - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
    - Ensure compatibility with hi2temp and make necessary adjustments.
  - **Week 6: (July 7 – July 13)**
    - Conduct comprehensive testing on all newly implemented functionalities.
    - Verify that nothing breaks in the existing codebase.
  - **Week 7: (July 14 – July 20)**
    - Focus on extensive testing to improve code coverage and ensure stability.
    - Address edge cases and unexpected scenarios.
  - **Week 8: (July 21 – July 27)**
    - Identify any remaining bottlenecks in the pipeline.
    - Optimize slow components using Numba or CuPy wherever necessary.
  - **Week 9: (July 28 – Aug 5)**
    - Add detailed docstrings and update documentation for all changes.
    - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

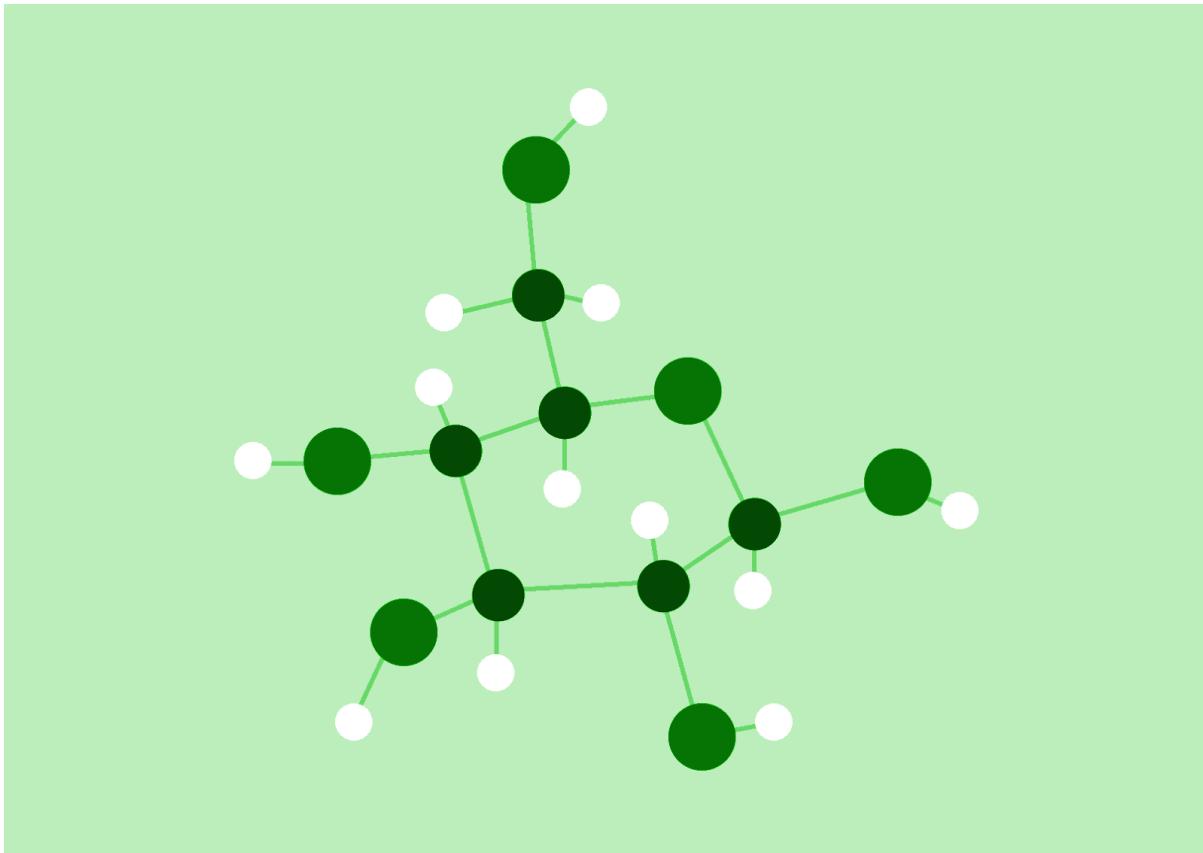
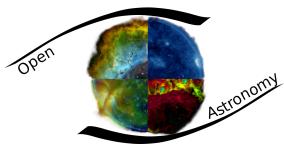
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

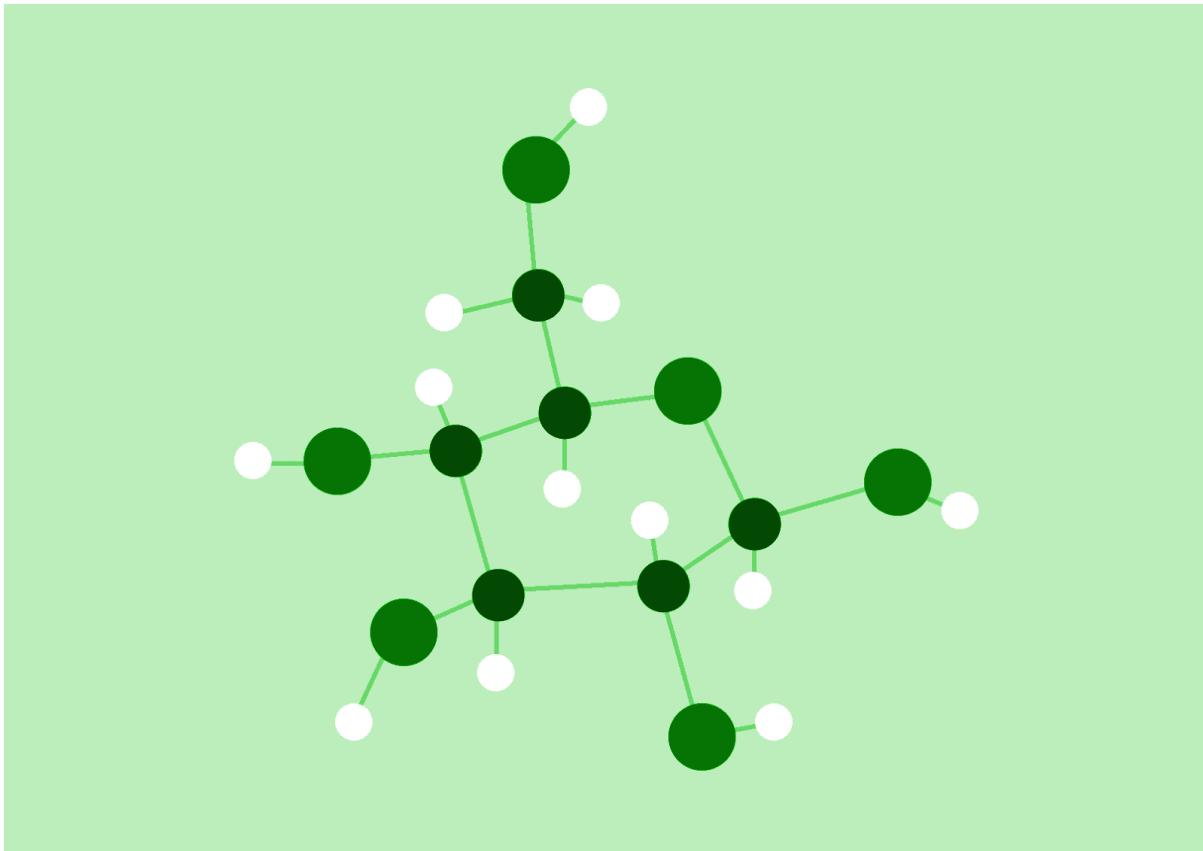
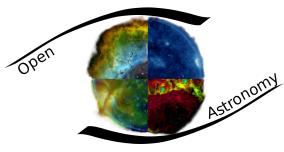
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

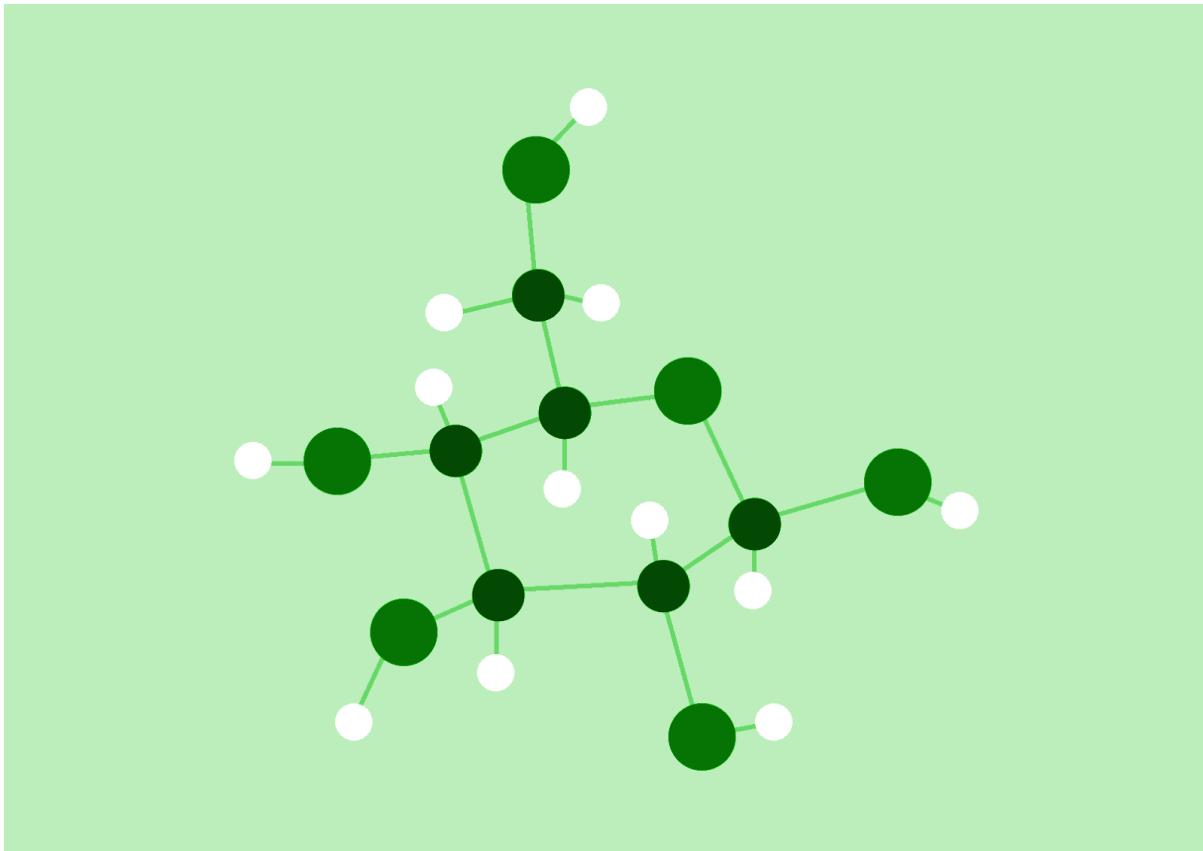
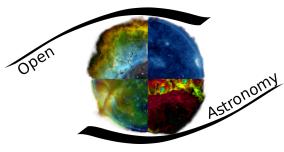
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

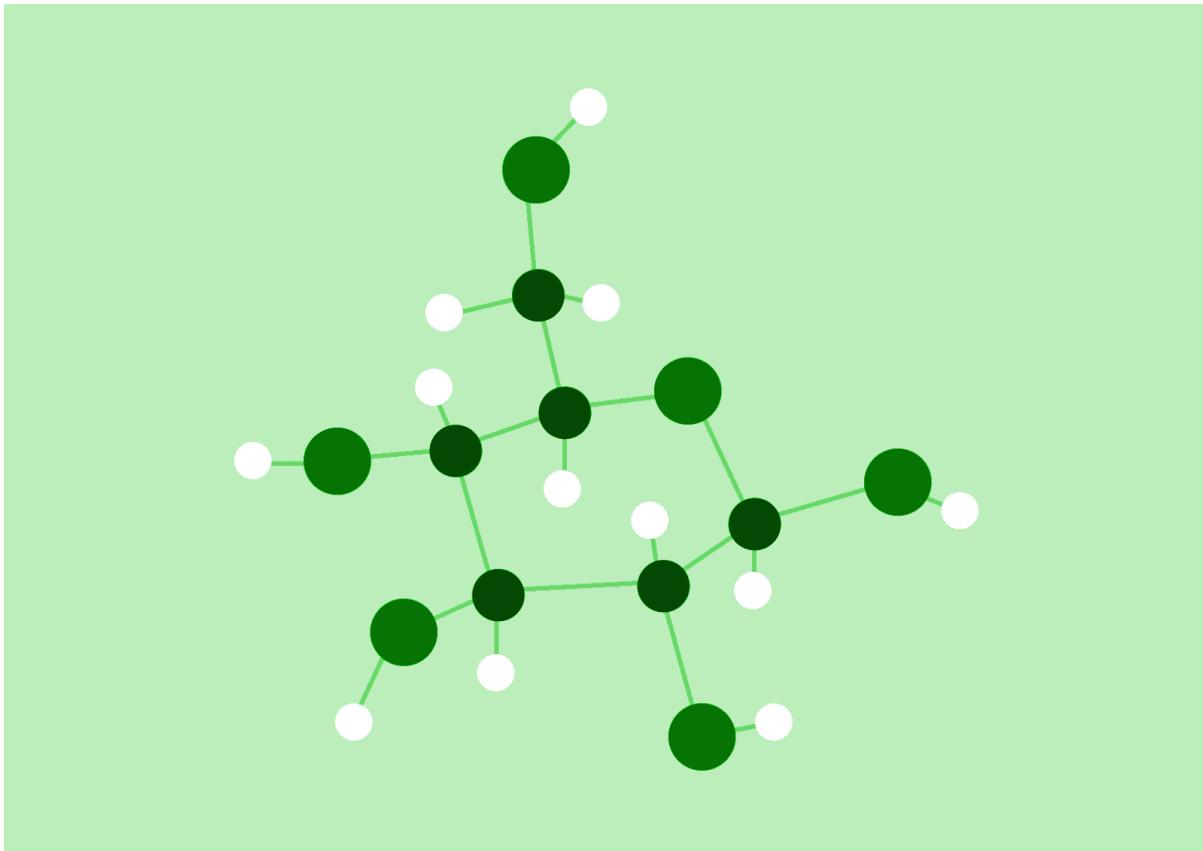
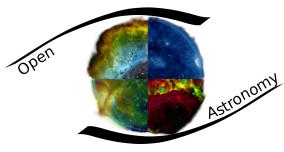
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



Radis

Fast parsing of large databases and  
execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
  - Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
    - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
    - Ensure compatibility with hi2temp and make necessary adjustments.
  - **Week 6: (July 7 – July 13)**
    - Conduct comprehensive testing on all newly implemented functionalities.
    - Verify that nothing breaks in the existing codebase.
  - **Week 7: (July 14 – July 20)**
    - Focus on extensive testing to improve code coverage and ensure stability.
    - Address edge cases and unexpected scenarios.
  - **Week 8: (July 21 – July 27)**
    - Identify any remaining bottlenecks in the pipeline.
    - Optimize slow components using Numba or CuPy wherever necessary.
  - **Week 9: (July 28 – Aug 5)**
    - Add detailed docstrings and update documentation for all changes.
    - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

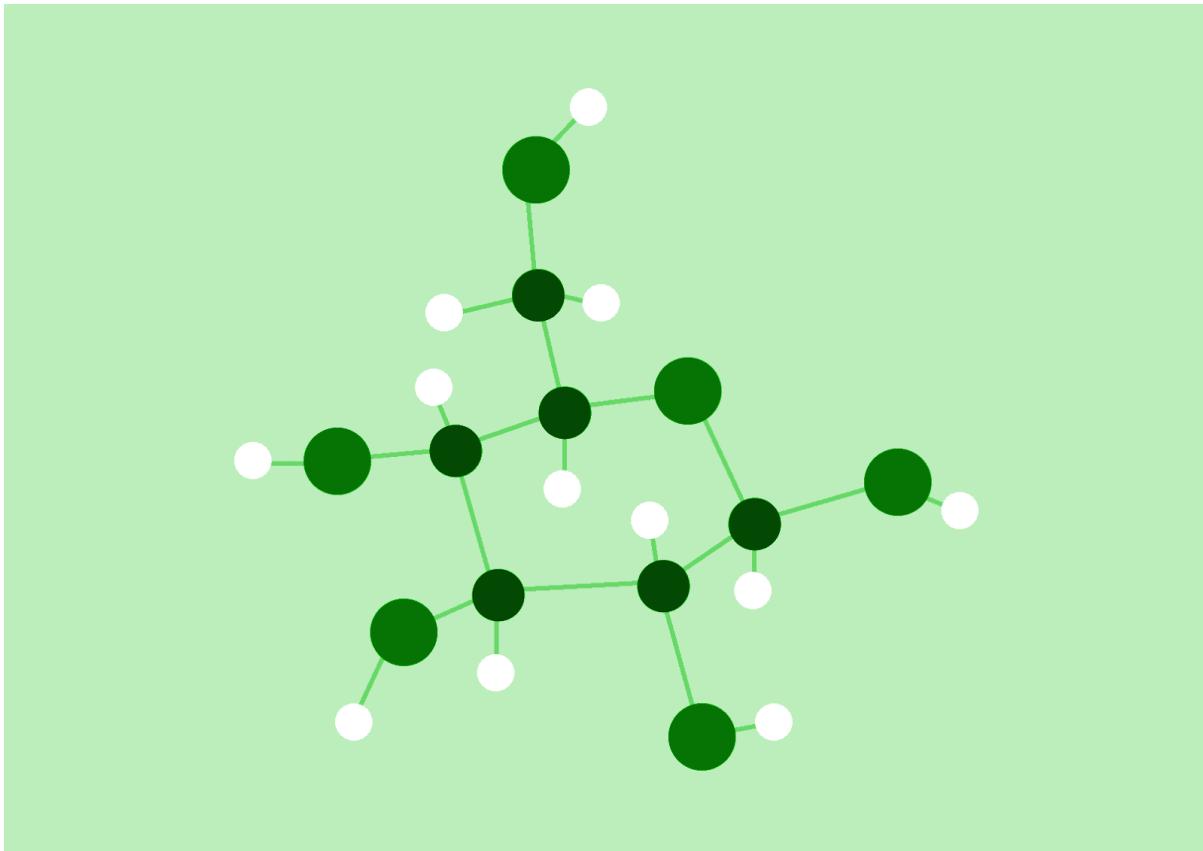
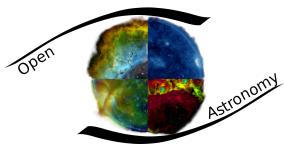
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



Radis

Fast parsing of large databases and  
execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

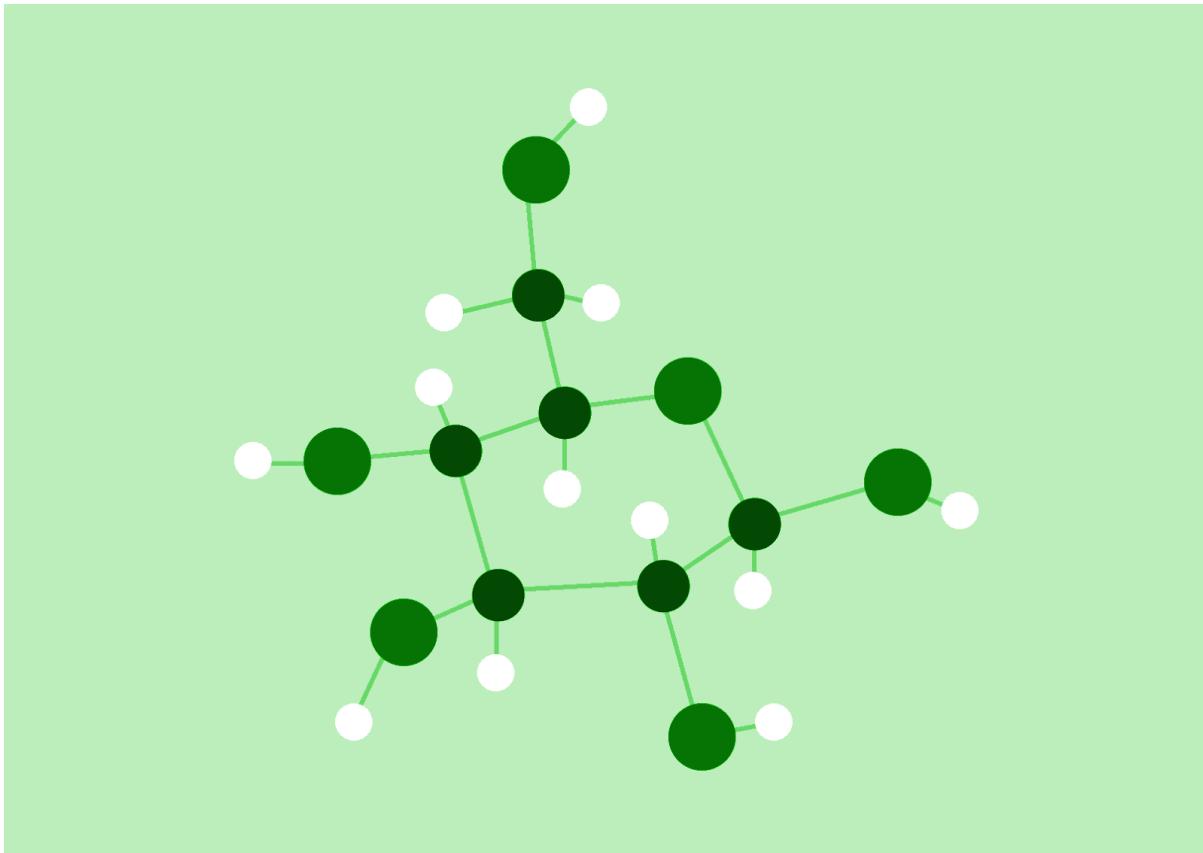
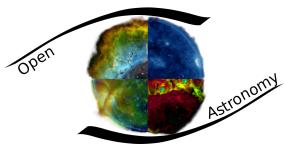
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

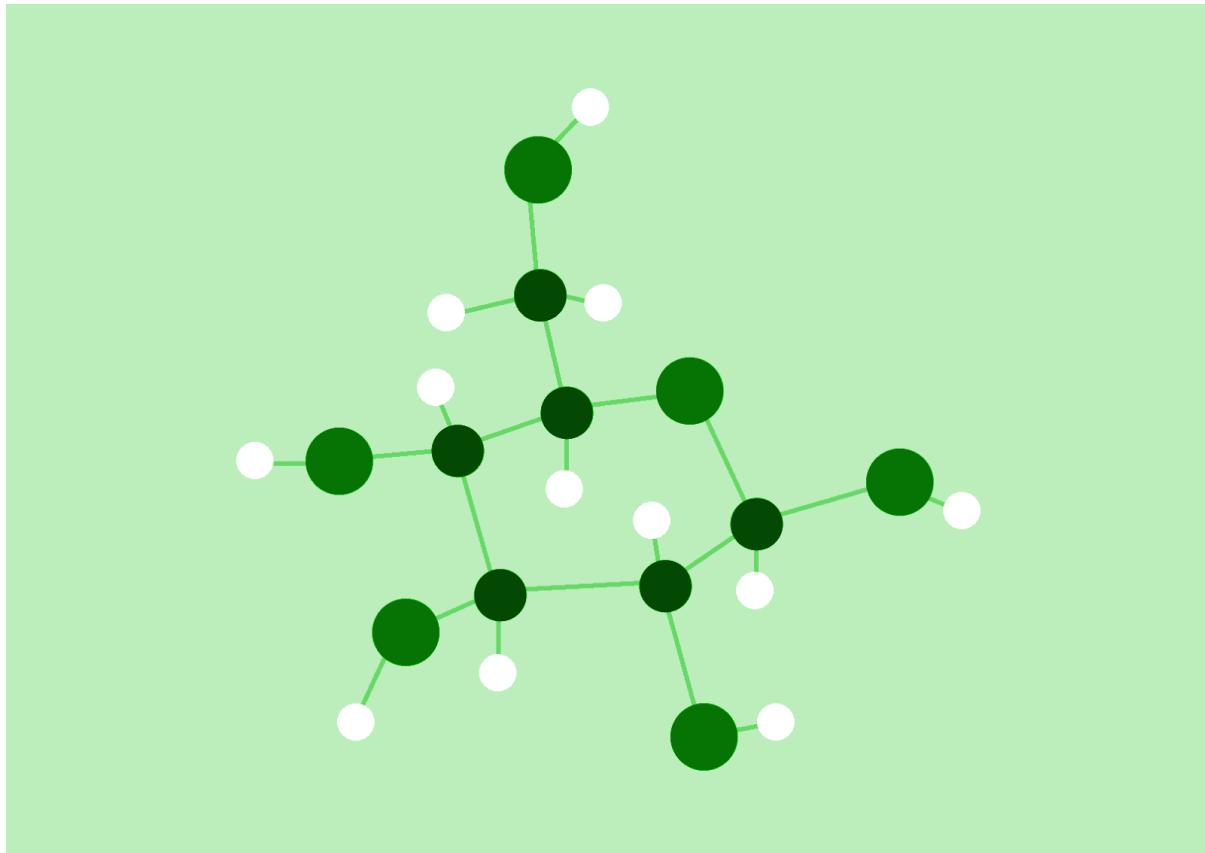
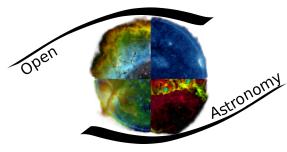
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

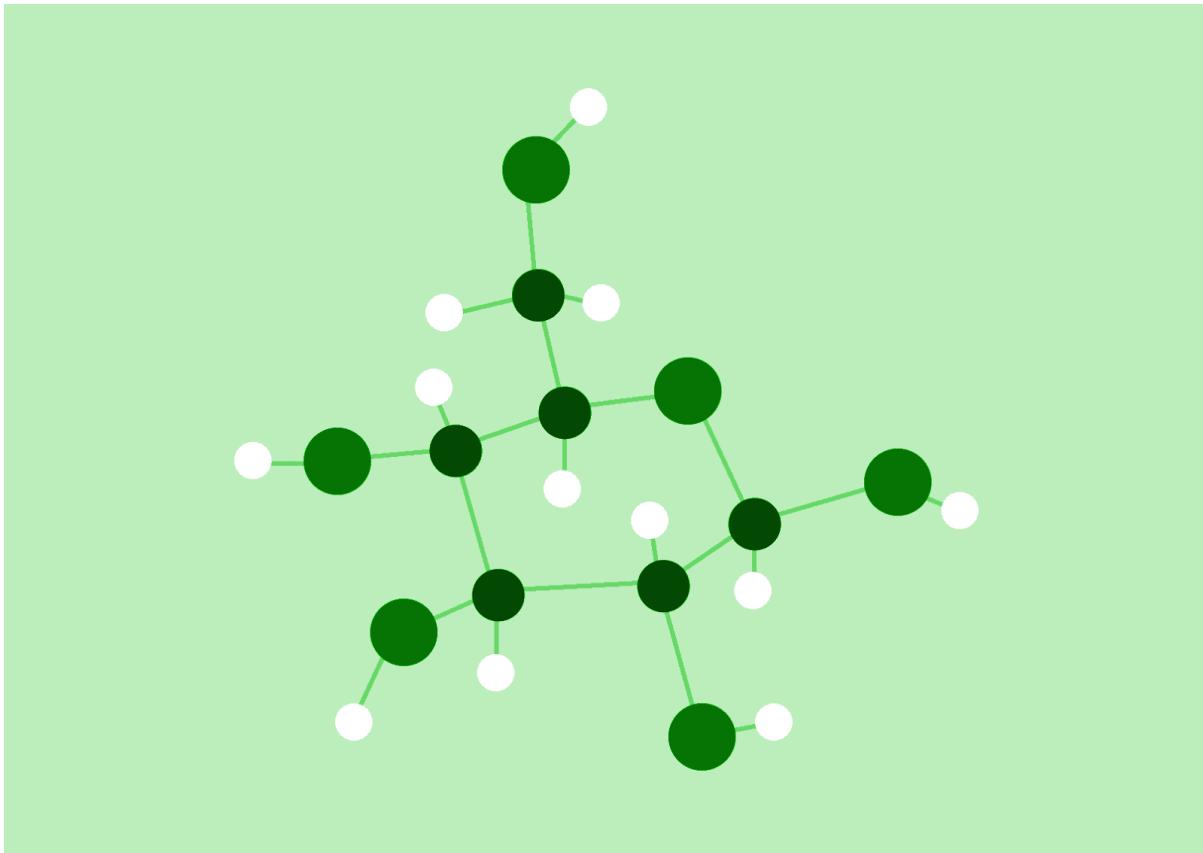
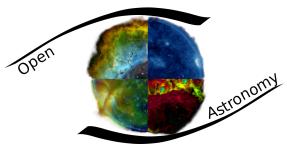
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
  - Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
    - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
    - Ensure compatibility with hi2temp and make necessary adjustments.
  - **Week 6: (July 7 – July 13)**
    - Conduct comprehensive testing on all newly implemented functionalities.
    - Verify that nothing breaks in the existing codebase.
  - **Week 7: (July 14 – July 20)**
    - Focus on extensive testing to improve code coverage and ensure stability.
    - Address edge cases and unexpected scenarios.
  - **Week 8: (July 21 – July 27)**
    - Identify any remaining bottlenecks in the pipeline.
    - Optimize slow components using Numba or CuPy wherever necessary.
  - **Week 9: (July 28 – Aug 5)**
    - Add detailed docstrings and update documentation for all changes.
    - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

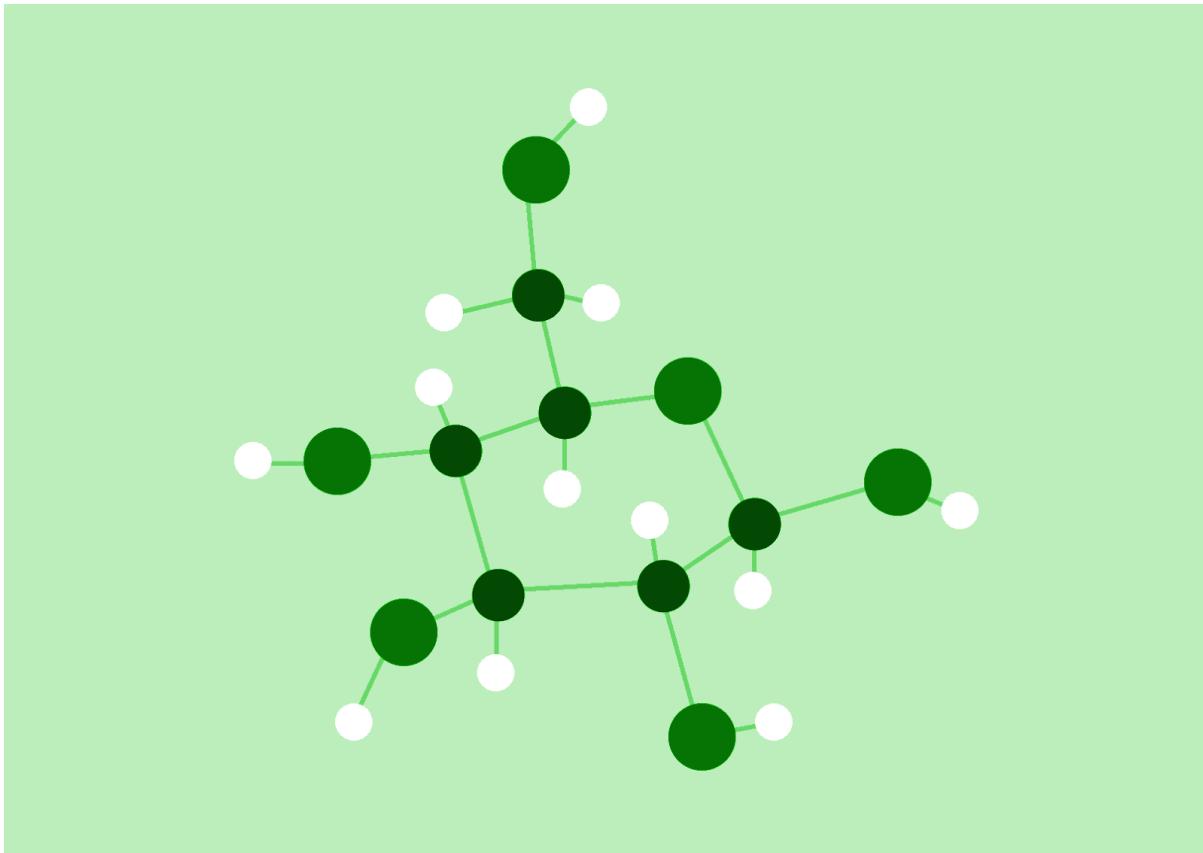
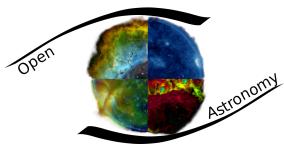
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

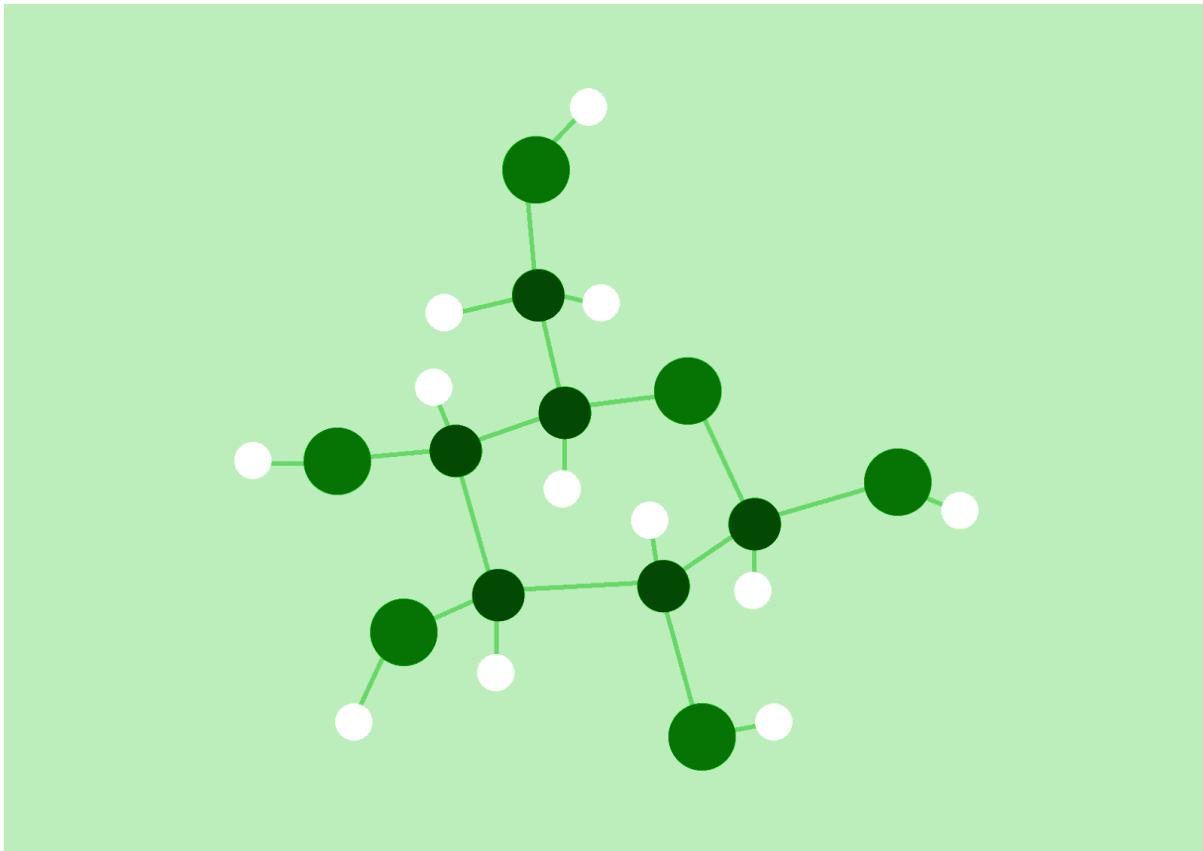
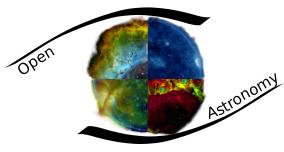
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

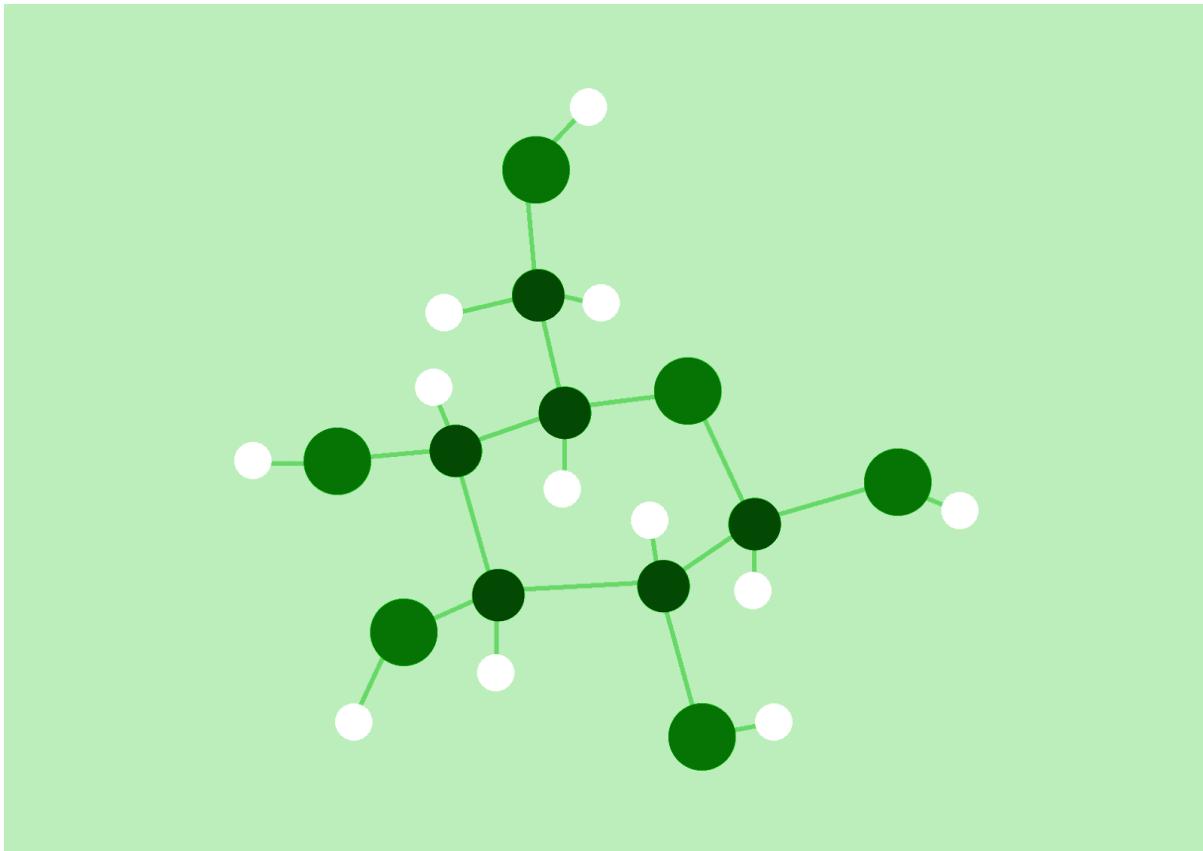
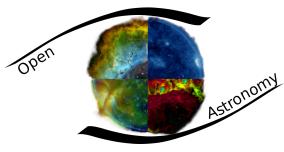
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

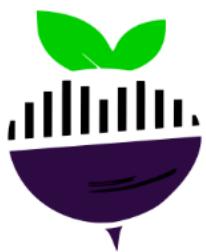
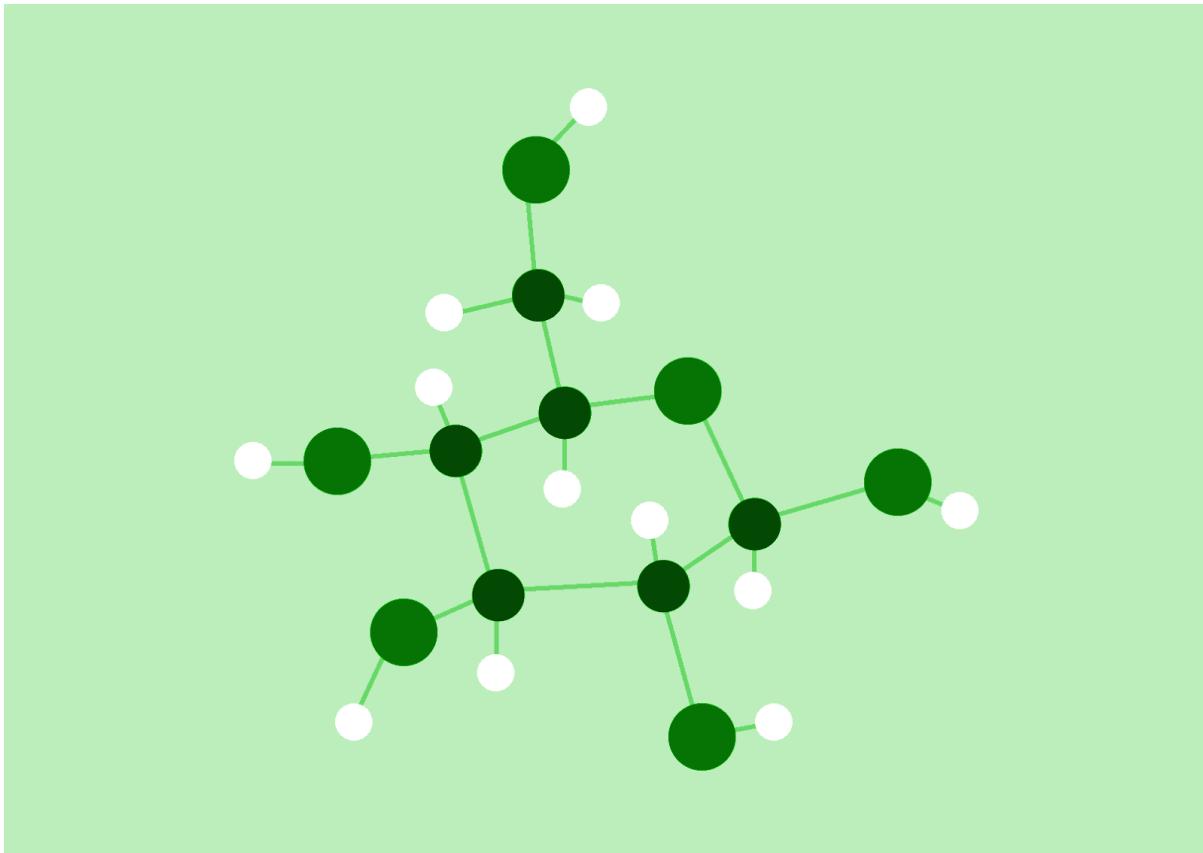
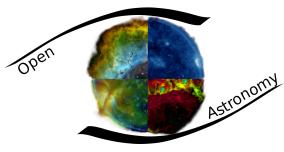
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

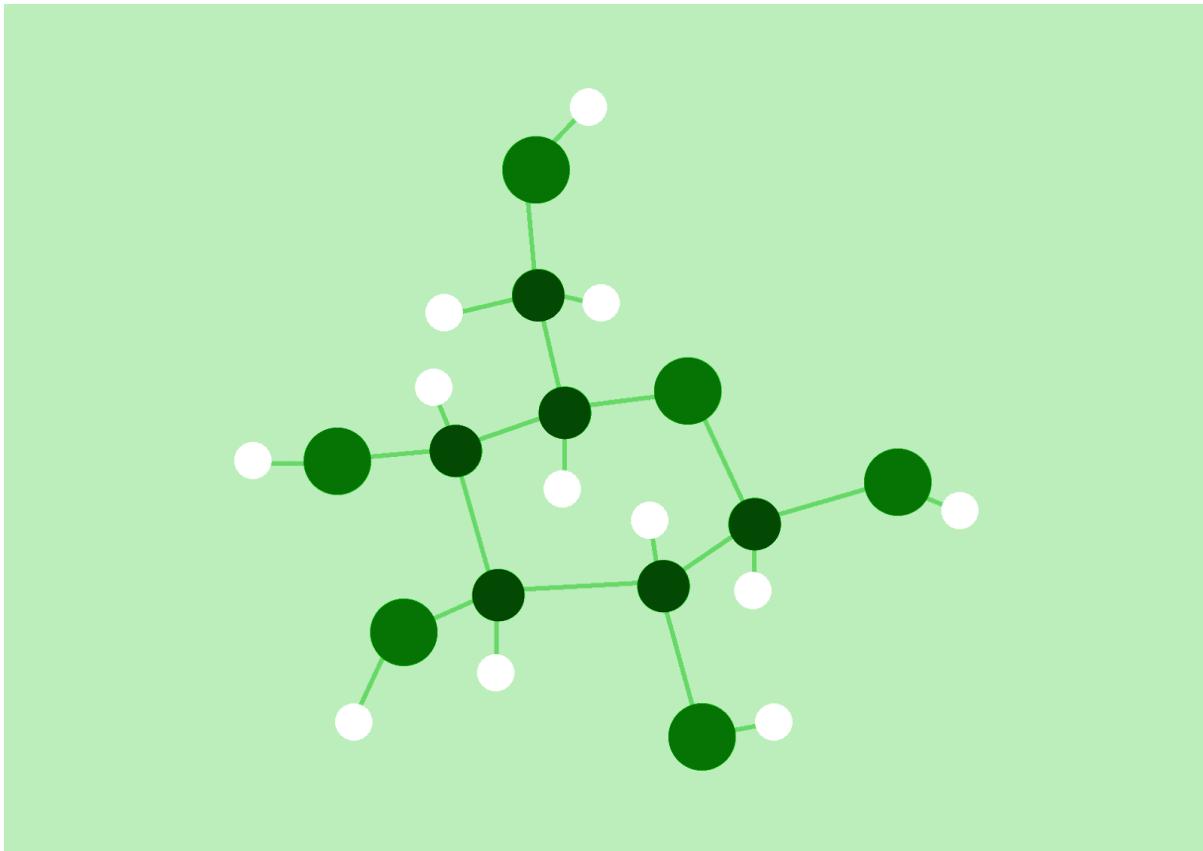
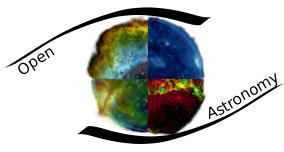
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



Radis

Fast parsing of large databases and  
execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

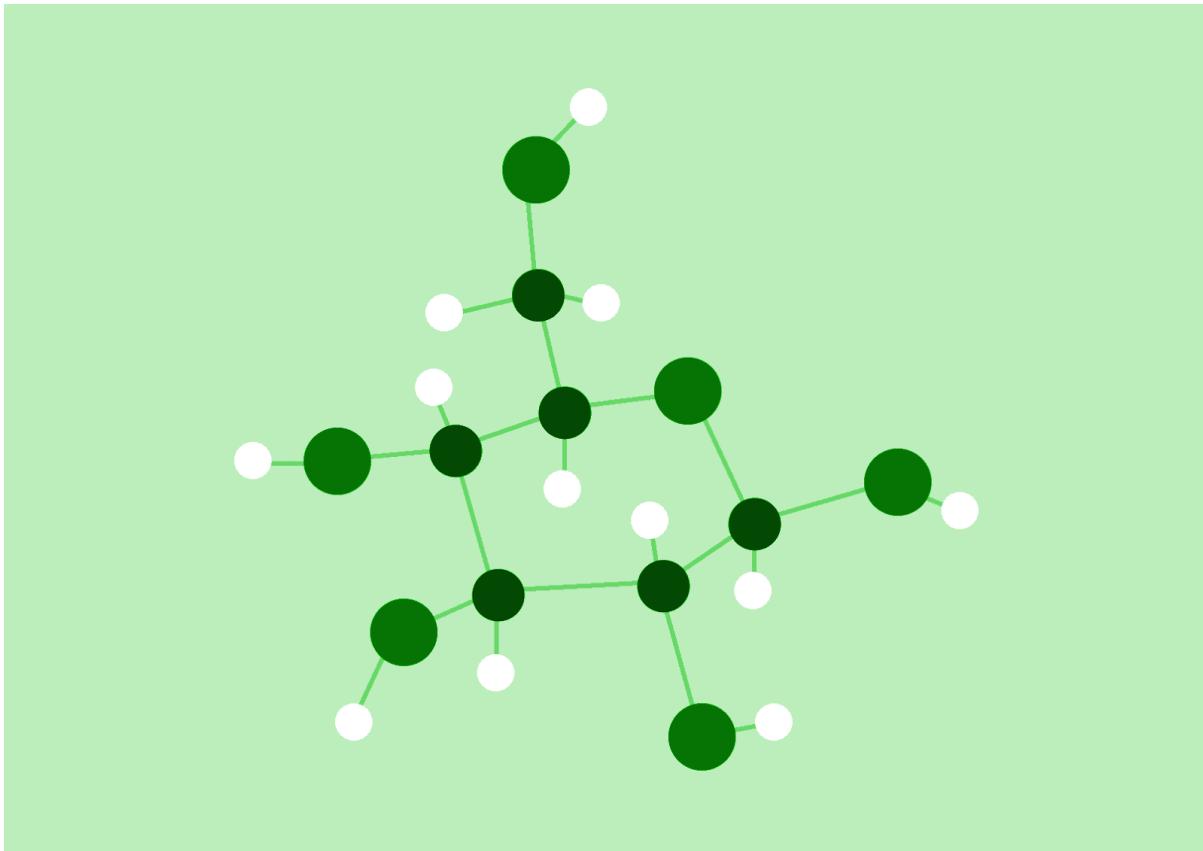
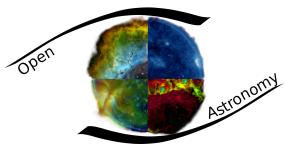
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

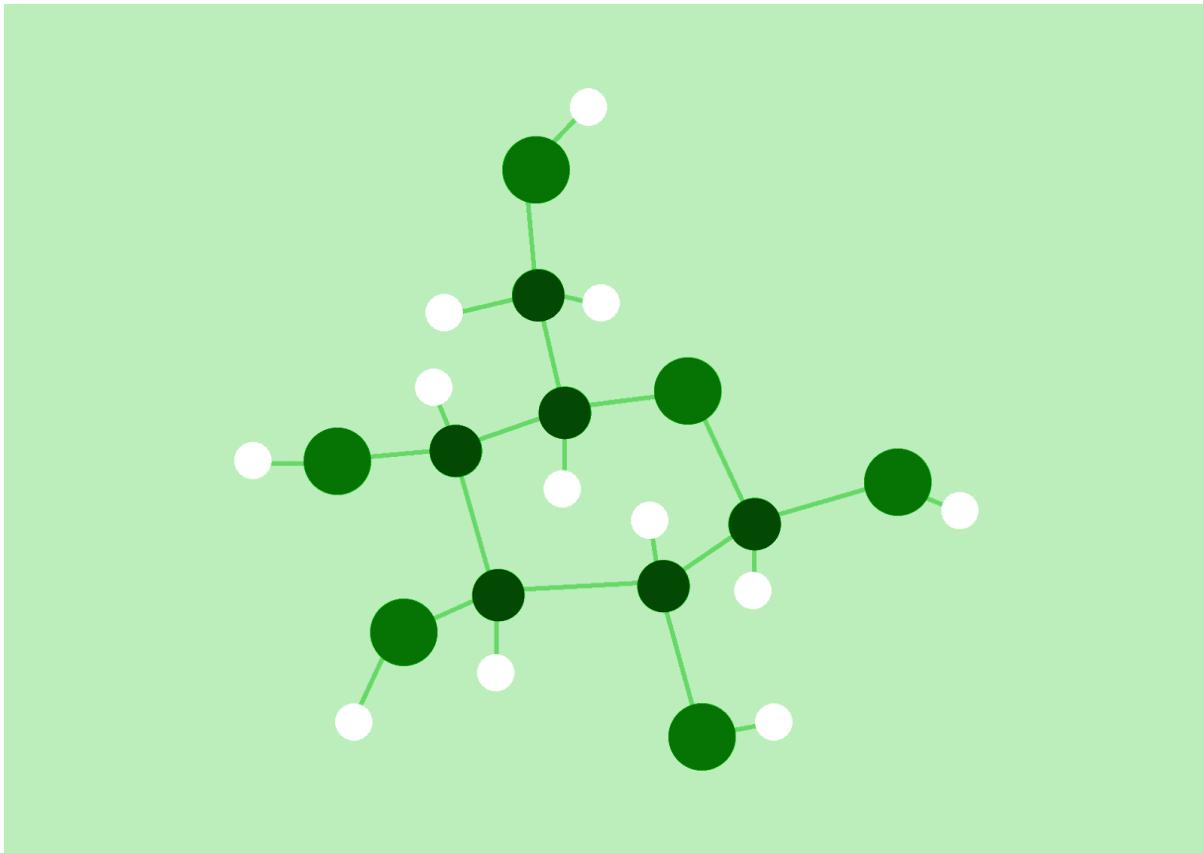
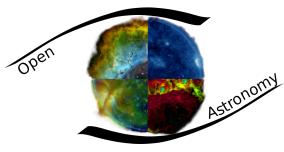
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
  - Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
    - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
    - Ensure compatibility with hi2temp and make necessary adjustments.
  - **Week 6: (July 7 – July 13)**
    - Conduct comprehensive testing on all newly implemented functionalities.
    - Verify that nothing breaks in the existing codebase.
  - **Week 7: (July 14 – July 20)**
    - Focus on extensive testing to improve code coverage and ensure stability.
    - Address edge cases and unexpected scenarios.
  - **Week 8: (July 21 – July 27)**
    - Identify any remaining bottlenecks in the pipeline.
    - Optimize slow components using Numba or CuPy wherever necessary.
  - **Week 9: (July 28 – Aug 5)**
    - Add detailed docstrings and update documentation for all changes.
    - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

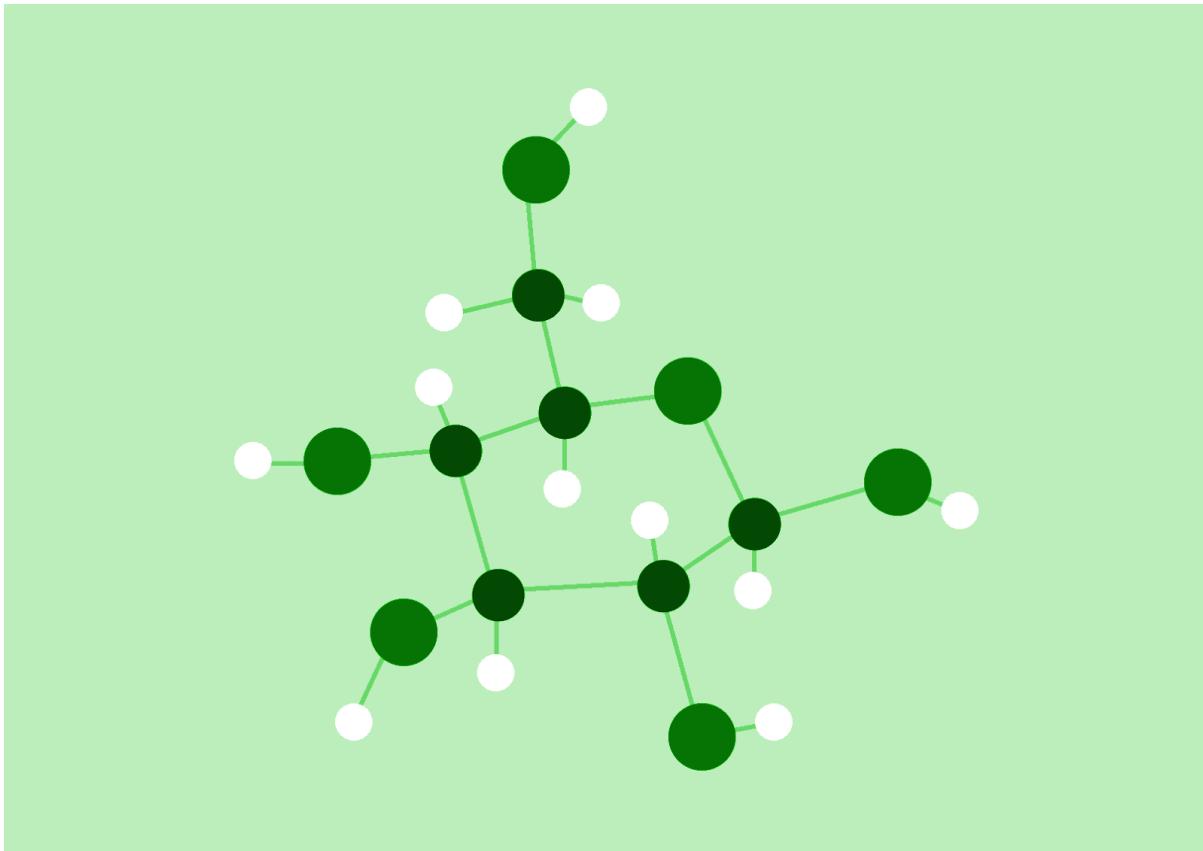
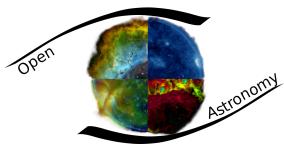
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
  - Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
    - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
    - Ensure compatibility with hi2temp and make necessary adjustments.
  - **Week 6: (July 7 – July 13)**
    - Conduct comprehensive testing on all newly implemented functionalities.
    - Verify that nothing breaks in the existing codebase.
  - **Week 7: (July 14 – July 20)**
    - Focus on extensive testing to improve code coverage and ensure stability.
    - Address edge cases and unexpected scenarios.
  - **Week 8: (July 21 – July 27)**
    - Identify any remaining bottlenecks in the pipeline.
    - Optimize slow components using Numba or CuPy wherever necessary.
  - **Week 9: (July 28 – Aug 5)**
    - Add detailed docstrings and update documentation for all changes.
    - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

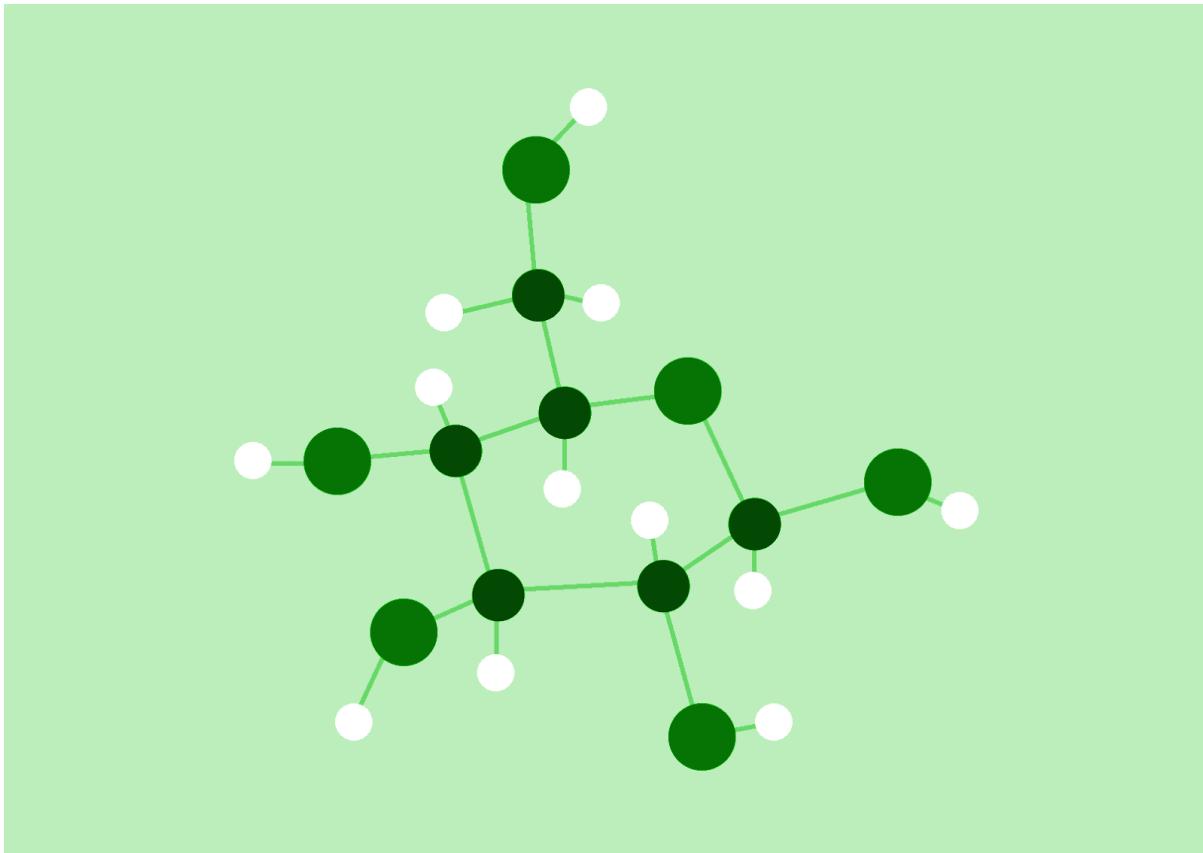
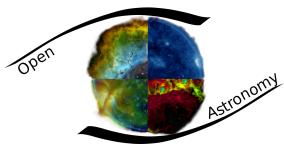
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
  - Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
    - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
    - Ensure compatibility with hi2temp and make necessary adjustments.
  - **Week 6: (July 7 – July 13)**
    - Conduct comprehensive testing on all newly implemented functionalities.
    - Verify that nothing breaks in the existing codebase.
  - **Week 7: (July 14 – July 20)**
    - Focus on extensive testing to improve code coverage and ensure stability.
    - Address edge cases and unexpected scenarios.
  - **Week 8: (July 21 – July 27)**
    - Identify any remaining bottlenecks in the pipeline.
    - Optimize slow components using Numba or CuPy wherever necessary.
  - **Week 9: (July 28 – Aug 5)**
    - Add detailed docstrings and update documentation for all changes.
    - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

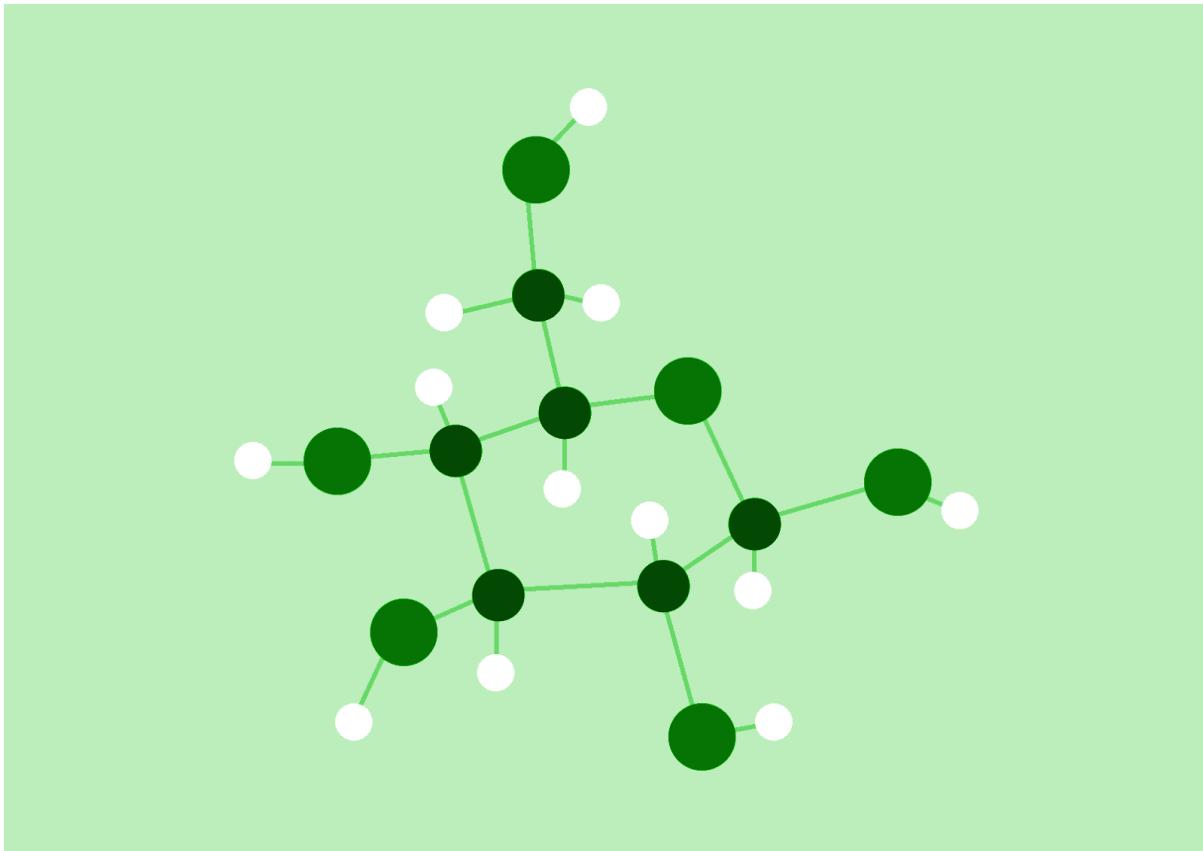
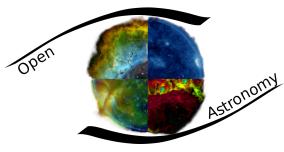
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
  - Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
    - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
    - Ensure compatibility with hi2temp and make necessary adjustments.
  - **Week 6: (July 7 – July 13)**
    - Conduct comprehensive testing on all newly implemented functionalities.
    - Verify that nothing breaks in the existing codebase.
  - **Week 7: (July 14 – July 20)**
    - Focus on extensive testing to improve code coverage and ensure stability.
    - Address edge cases and unexpected scenarios.
  - **Week 8: (July 21 – July 27)**
    - Identify any remaining bottlenecks in the pipeline.
    - Optimize slow components using Numba or CuPy wherever necessary.
  - **Week 9: (July 28 – Aug 5)**
    - Add detailed docstrings and update documentation for all changes.
    - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

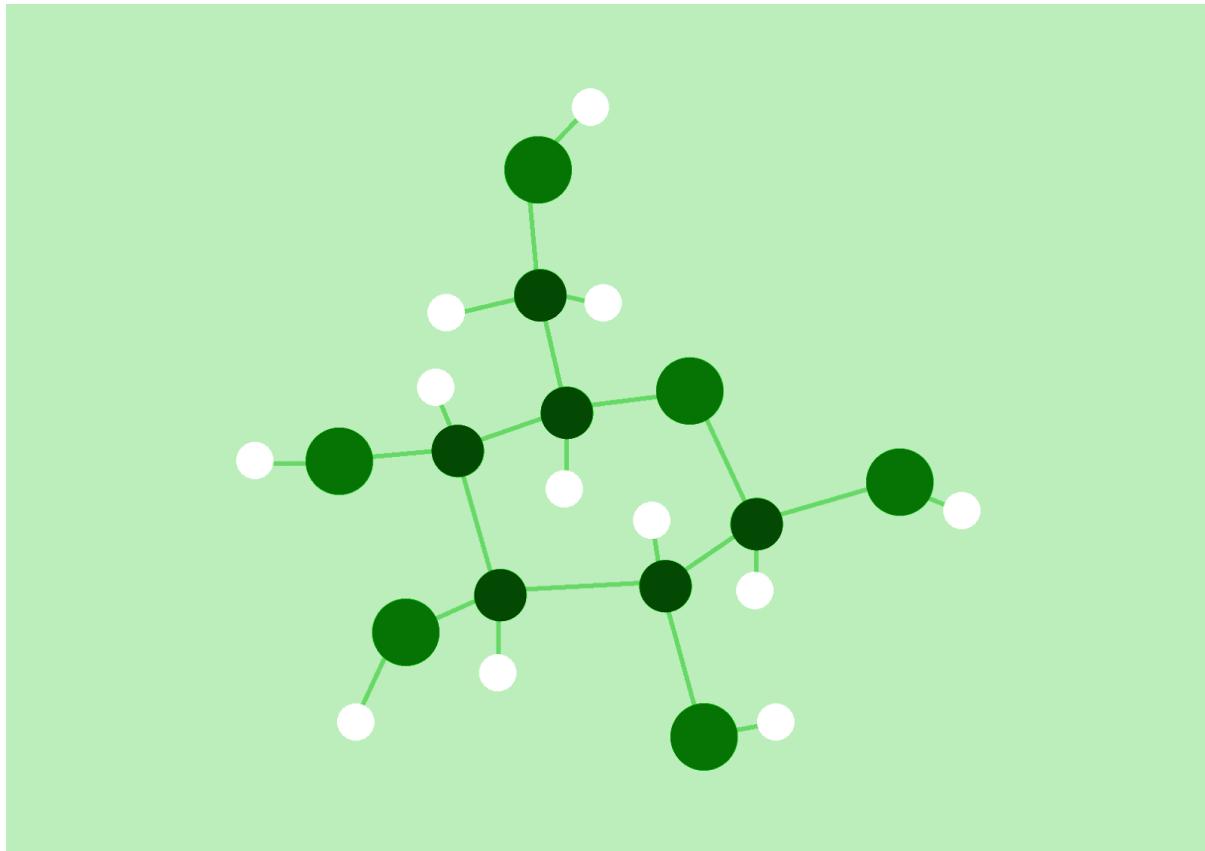
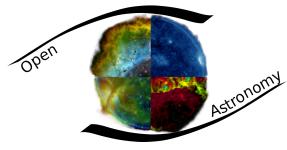
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

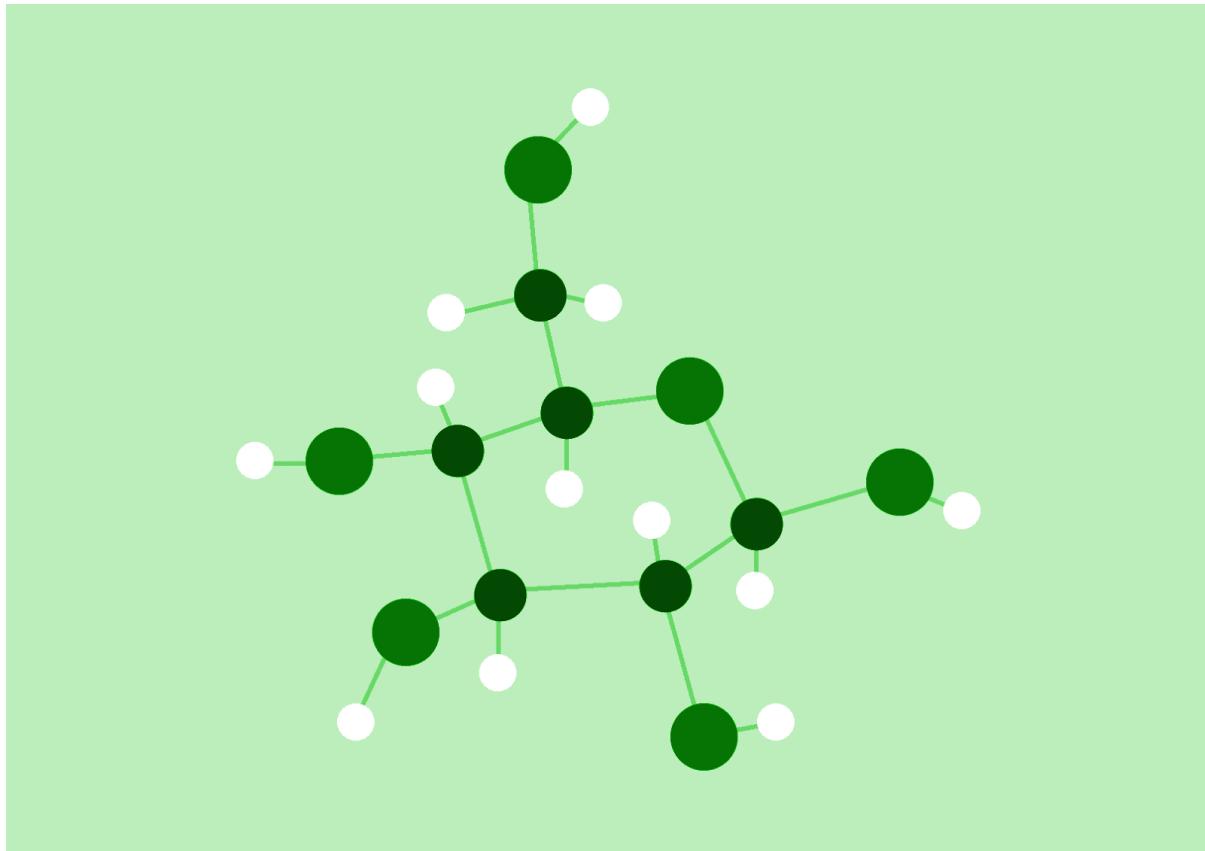
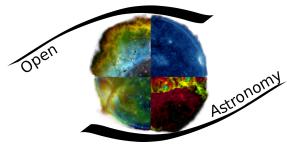
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

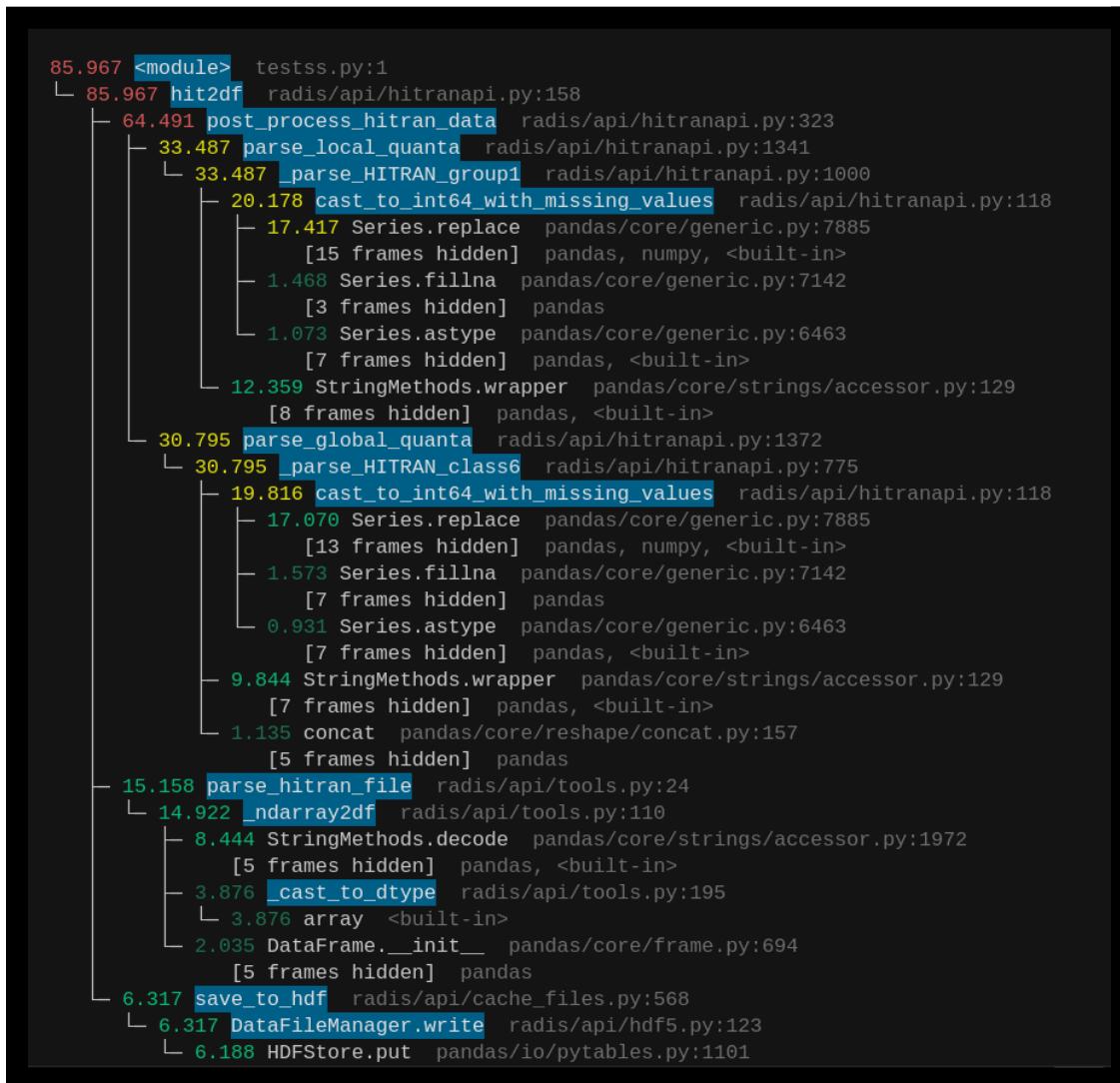
In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

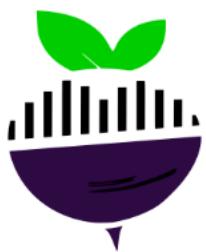
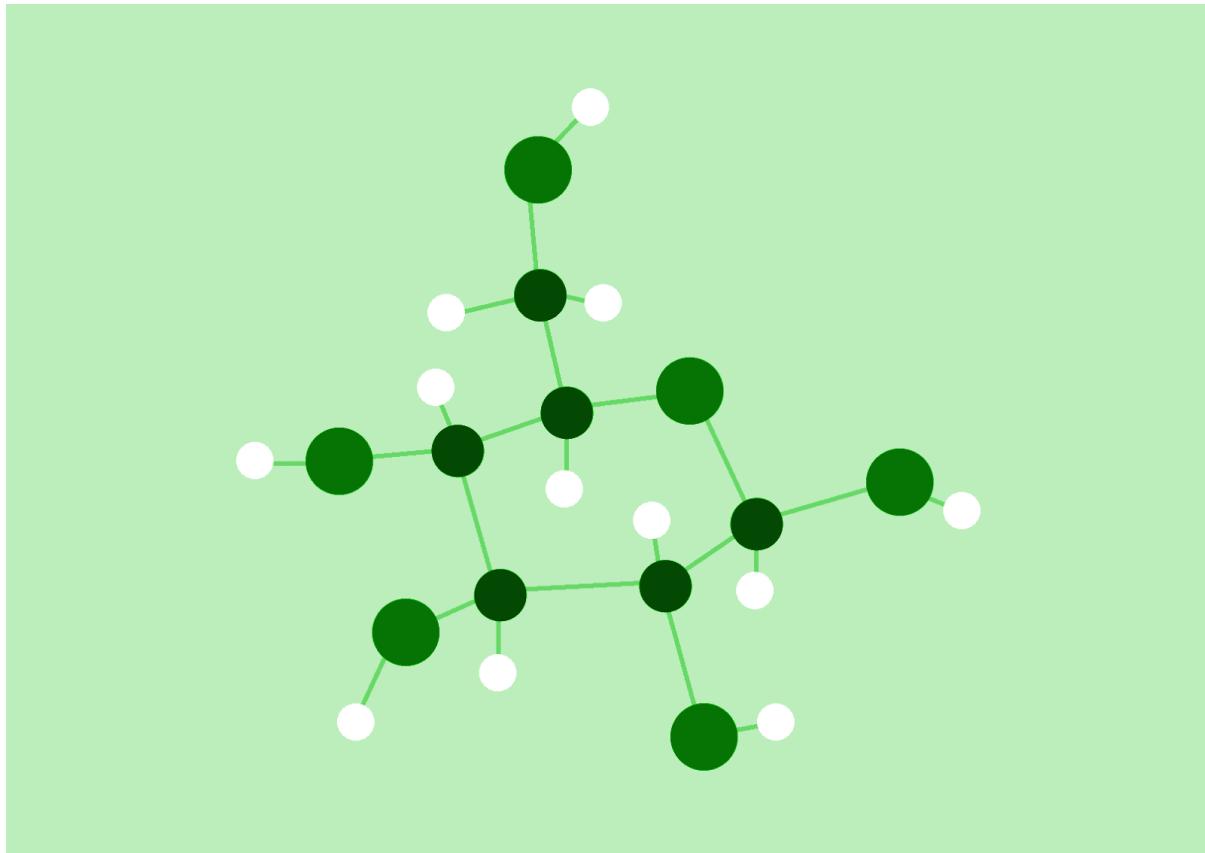
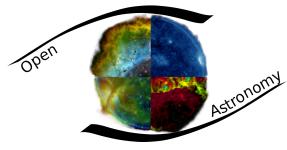
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
  - Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
    - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
    - Ensure compatibility with hi2temp and make necessary adjustments.
  - **Week 6: (July 7 – July 13)**
    - Conduct comprehensive testing on all newly implemented functionalities.
    - Verify that nothing breaks in the existing codebase.
  - **Week 7: (July 14 – July 20)**
    - Focus on extensive testing to improve code coverage and ensure stability.
    - Address edge cases and unexpected scenarios.
  - **Week 8: (July 21 – July 27)**
    - Identify any remaining bottlenecks in the pipeline.
    - Optimize slow components using Numba or CuPy wherever necessary.
  - **Week 9: (July 28 – Aug 5)**
    - Add detailed docstrings and update documentation for all changes.
    - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

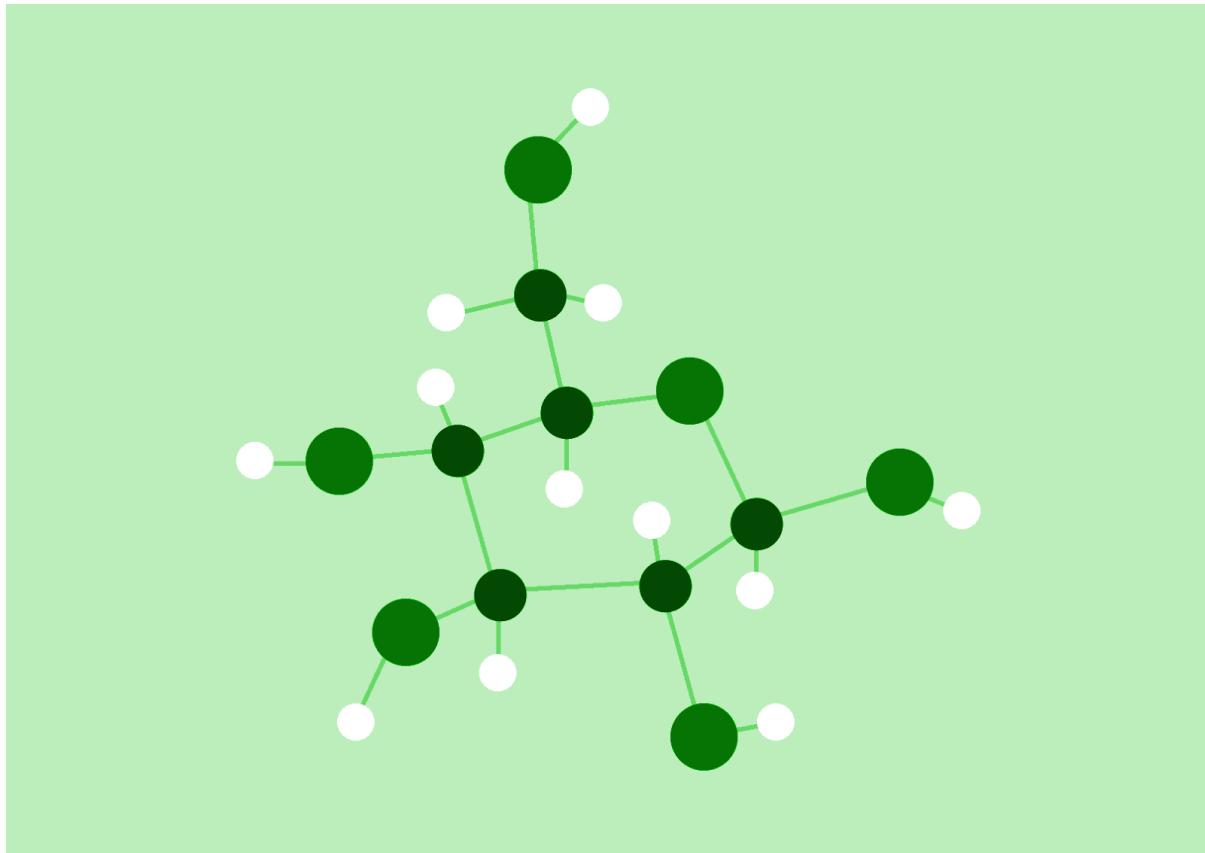
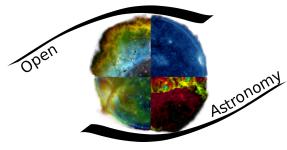
I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.



## Radis

Fast parsing of large databases and execution bottlenecks

---

Organisation: OpenAstronomy

Sub Organisation: Radis

Mentors: dcmvdbekerom minouHub TranHuuNhatHuy By: Pratham Hole

## Contact Information:

**Name:** Pratham Hole

**E-mail:** [prathamhole@gmail.com](mailto:prathamhole@gmail.com) [lcb2023003@iiitl.ac.in](mailto:lcb2023003@iiitl.ac.in)

**GitHub:** <https://github.com/Prtm2110>

**LinkedIn:** <https://www.linkedin.com/in/pratham-hole>

**University:** Indian Institute of Information Technology, Lucknow (<https://iiitl.ac.in/>)

**Location:** India / IST, GMT +05:30

**Contact Number:** +91 - 95526 38515

## Project Proposal:

### Abstract:

The RADIS code was developed for the characterization of plasmas, flames and atmospheres. High-temperature spectral calculations require resolving the shape of tens of millions of lines, which is the usual performance bottleneck. RADIS implements a new algorithm to compute these lineshapes and is already one of the fastest line-by-line spectral codes available.

With more and more lines added to the databases, some files grew considerably in size. To be as fast as possible, RADIS converts compressed databases into the HDF5 format. This conversion also increases the size of the files which can take hours to be written in the hard drive (2-3 hours for HITEMP CO<sub>2</sub> in its 2025 version- [link](#)). The objective of this project is to accelerate the parsing to HDF5 files.

### Technical Details:

HITRAN data is provided in .par files that are either bz2 or zip compressed. These files must be uncompressed, and the primary goal is to parse the parameter (.par) files and convert them into a DataFrame (using Pandas and Veax) and subsequently save the data in HDF5 format.

The key objectives of this project are as follows:

- Identifying the bottlenecks in the entire file fetching and parsing process from the HITRAN Database.
- One well-known bottleneck is parsing parameter (.par) files into HDF5, this can be addressed using the existing C++ parsing code provided by @dcmvdbekerom ([link](#))
- Optimising functions called during the execution of [hit2df](#) with the help of numba.

- Documenting a review of the current bottlenecks in parsing and computing spectra, with working examples for small ( $\sim 1 \text{ cm}^{-1}$ ) and large ( $\sim 100 \text{ cm}^{-1}$ ) spectra.

Following is a brief overview of the project:

## Part 1: Ability to Partially Download from the HITEMP Database

The HITRAN database provides spectroscopic data for CO<sub>2</sub> in BZ2 compressed formats. Currently, the implementation fetches and downloads the entire database, which can be highly inefficient. The CO<sub>2</sub> dataset is 6.4 GB in a single, highly compressed BZ2 file. Downloading and processing this large file requires significant bandwidth and time, even if only a small wavenumber range is needed.

To address this, I shall add the ability for partial downloads based on a specified wavenumber range. This would allow users to retrieve only the necessary spectral data, significantly reducing download time, storage requirements, and parsing overhead. Implementing this feature will enhance efficiency and make data access more practical for various applications.

This process can be achieved using the [code](#) provided by [@dcmvdbekerom](#). It involves executing `gen_bit_shifts.py` and `find_block_header_offsets.py`, which take a few minutes to run. After that, `partial_decompression_into_bz2.py` is used to slice the `.bz2` file into arbitrary blocks, as mentioned [here](#). To enable this the necessary API will be implemented and refactored within the [DatabaseManager](#) and [HITEMPDatabaseManager](#) classes.

One possible enhancement is enabling parallel downloads for multiple files in [HITRANDatabaseManager](#) (specifically for H<sub>2</sub>O) using either [ThreadPoolExecutor](#) or [asyncio + aiohttp](#). This would significantly improve performance by reducing download time, especially for large datasets. Additionally, we can optimize file extraction by leveraging multithreading for unzipping the downloaded files. This approach would ensure that both downloading and processing happen concurrently, leading to faster data availability and improved efficiency in handling large datasets.

## Part 2: Constructing a Highly Efficient Parsing Backend

We can construct the parsing backend in the following ways, and after benchmarking different approaches, the best method will be chosen for actual implementation in Radis.

**Option 1.** Implement the C++/SIMD code with the help of `Pybind11` so that it can be used as a Python module. This module will need to be pre-built for every Python version, platform, and architecture, which can be managed using `cibuildwheels`. The code should account for the fact that different HITRAN molecules have varying column structures; therefore, it must create the corresponding binary files accordingly. The output of this parser is binary files in `.dat` format that can be read using NumPy's `fromfile` function.

**Option 2.** Alternatively, rewrite the C++/SIMD code in Python using PySIMD (<https://github.com/jweinstl/pysimd>) to improve maintainability.

**Option 3.** The main bottleneck for parsing is Input/Output operations, which must be handled by the CPU, while other calculations can be performed on the GPU. Writing these computations in Python would make the implementation easier to maintain without requiring compilation. There are a few ways we can proceed (this can be decided either before submitting the proposal or during 'community bonding':

- Vulkan API (already implemented in RADIS, preferred) or OpenCL (alternative).
- Using CUDA based approaches such as Numba and Cupy.

### Part 3: Forming a DataFrame and Storing in HDF5 Format

After parsing the database, the next step is to construct a DataFrame using either pandas or Vaex and then convert and store it in an HDF5 file using [\(save\\_to\\_hdf\)](#). HDF5 allows for faster data retrieval and eliminates the need for repeated parsing, significantly improving efficiency. This approach is similar to what [hit2df](#) does.

### Part 4: Implementation of the New Fast Parsing Backend in RADIS

In this phase, the focus is on integrating a new, high-speed parsing mechanism into RADIS while cleaning up the existing code. The plan involves refactoring the `parse_hitran_file` function in `tools.py` by embedding the selected parsing strategy and removing redundant functionalities. Additionally, efforts will be made to ensure that the updated parser works with `hi2temp` by fine-tuning compatibility, optimizing memory usage, and improving processing speed. Detailed documentation and benchmarking tests will be carried out throughout the process to verify the changes and measure performance improvements.

## Part 5: Identifying Additional Bottlenecks and Benchmarking Against Existing Parsing

After implementing the new hit2df parsing backend, the next step is to profile its performance to identify further bottlenecks. This includes issues related to extracting BZ2 files from HITRAN.

These bottlenecks will be addressed using Numba or by rewriting critical sections in NumPy or similar optimized libraries.

Currently, profiling on the file `01_4500-5000_HITEMP2010.par` ([link](#)) (a 459 MB file after extraction from the ZIP, for the CO<sub>2</sub> HITEMP database) indicates that the `post_process_hitran_data` function is a major bottleneck. This function will be replaced by the new parsing backend developed in Part 3, significantly improving performance.



## Part 6: Document All the Changes, Add Gallery Examples, and Develop a Testing Framework

Document every aspect of the code architecture with a comprehensive developer guide that clarifies any ambiguities that may arise in the future. Additionally, develop and run tests to ensure the system functions correctly across all target operating systems, and include gallery examples to demonstrate usage.

## Schedule of Deliverables:

### Community bonding:

Engage in detailed discussions with mentors and the community to refine the overall concept, clearly define what is required to implement a new feature or enhance existing ones, and thoroughly review the proposed approach with mentors to avoid early-stage missteps. Additionally, familiarize myself with the project's expectations and key requirements.

Below is a more structured, week-by-week timeline for the 9-week GSoC coding period to help ensure steady progress:

- **Week 1: (June 2 – June 8)**
  - Implement the backend for partial downloads directly from compressed files.
  - Optimize the implementation using Numba, NumPy, or CuPy for performance gains.
- **Week 2: (June 9 – June 15)**
  - Remove the existing code and integrate the new partial download mechanism.
  - Ensure everything is working efficiently and is well-tested.
- **Week 3: (June 16 – June 22)**
  - Develop a parsing backend leveraging NumPy for speed and efficiency.
  - Develop parsing backed with help of C++ SIMD
- **Week 4: (June 23 – June 29)**
  - Explore CUDA-based parsing to take advantage of GPU acceleration.

- Explore vulkan API or openCL -based parsing to take advantage of GPU acceleration.
- Benchmark all three parsing approaches (vulkan, CUDA, and C++ SIMD) and, with mentors, finalize the best option.
- **Week 5: (June 30 – July 6)**
  - Refactor parse\_hitran\_file from tools.py, integrating the newly selected parsing mechanism.
  - Ensure compatibility with hi2temp and make necessary adjustments.
- **Week 6: (July 7 – July 13)**
  - Conduct comprehensive testing on all newly implemented functionalities.
  - Verify that nothing breaks in the existing codebase.
- **Week 7: (July 14 – July 20)**
  - Focus on extensive testing to improve code coverage and ensure stability.
  - Address edge cases and unexpected scenarios.
- **Week 8: (July 21 – July 27)**
  - Identify any remaining bottlenecks in the pipeline.
  - Optimize slow components using Numba or CuPy wherever necessary.
- **Week 9: (July 28 – Aug 5)**
  - Add detailed docstrings and update documentation for all changes.
  - Compile a benchmark report, showcasing speed and performance improvements in fetching and parsing the database.

## After Summer of Code:

I plan to continue contributing to Radis, as I have been doing since February. My initial focus will be on completing any remaining goals from the project. Following that, I intend to actively maintain the code developed during the GSoC period and support the community by addressing issues and providing guidance on the usage of the new features.

## Development Experience:

I first started coding in my junior year of high school, trying to build simple games in Python. Later, in my freshman year of college, I began learning machine learning, which introduced me to the Pythonic open-source ecosystem. This led me to contribute to various open-source projects like SunPy (where I have been contributing for almost a year), sktime, cve-bin-tool, PyBaMM, and Radis.

Some of my open-source contributions are listed as follows:

### Radis:

- [#722](#) Fixes critical fetch\_hitemp bug
- [#756](#) Calculate Download Size in ExoMol and Warn Users for Large Downloads in ExoMol & HITEMP
- [#473](#) Ensure HITRAN Login is Compatible with Spyder with PyQt Support
- [#755](#) Allowing Caching in calc\_spectrum
- [#752](#) Adds different ways of calculating spectrum for multiple molecule example

### SunPy:

- [#8041](#) Adds ability to make superpixels masked if any of its constituent pixels is masked.
- [#7760](#) Adds get\_contours and supports "method" keyword argument to select between ContourPy and scikit-image
- [#7772](#) Adds reproject as an option for autoalign plotting of maps
- [#7844](#) GenericMap.draw\_contour() makes use of the plot\_settings dictionary
- [#7899](#) Adds show\_in\_notebook method for Interactive tables using itables
- [#7752](#) The RotatedSunFrame class now accepts an astropy.time.TimeDelta object for the duration parameter.

## Appendix:

### How much time will I be able to contribute to the project:

I am based in the GMT +05:30 time zone and will be able to dedicate approximately 30–35 hours per week to the project. In addition to my coding contributions, I plan to publish weekly blog updates and a Medium post summarizing the progress and key developments of my project. The official GSoC timeline aligns well with my university's summer break, providing me with sufficient uninterrupted time to focus on the project. Although my university resumes in mid-July, the initial weeks of the semester are generally light, with no exams, tests, or major assignments scheduled, ensuring that I can continue to dedicate substantial time and attention to completing the project within the expected timeframe.

### More about myself:

I am a sophomore at the Indian Institute of Information Technology, Lucknow (IIITL), majoring in Computer Science. I came across SunPy because it was Pythonic and related to physics, which interests me. I have been contributing to SunPy for more than half a year, and through it, I discovered RADIS. The supportive nature of the RADIS community made me stick with them.

During the early years of college, I worked on several projects, including a CLI tool titled *Vertex-CLI* ([link](#)) in Python and *Rabbit3D* ([link](#)), a rendering engine written in C++ with OpenGL as the backend.

I am excited about the opportunity to work with Radis on this project over the summer, as I believe I have the necessary skills and dedication to see it through to completion. I have already begun engaging with the community and familiarizing myself with the project's foundational concepts.

If selected, I am fully committed to giving my best effort to ensure the successful development and delivery of this project.

I look forward to the possibility of working with the Radis community and contributing meaningfully to the project.