Draft version December 5, 2018
Preprint typeset using LaTeX style AASTeX6 v. 1.0

MARVIN: A TOOLKIT FOR STREAMLINED ACCESS AND VISUALIZATION OF THE SDSS-IV MANGA DATA SET

Brian Cherinka[1], Brett H. Andrews[2], José Sánchez-Gallego[3], Joel Brownstein[4], María Argudo-Fernández[5,6], Michael Blanton[7], Kevin Bundy[8], Amy Jones[13], Karen Masters[10,11], David R. Law[1], Kate Rowlands[9], Anne-Marie Weijmans[12], Kyle Westfall[8], Renbin Yan[14]

[1]Space Telescope Science Institute, 3700 San Martin Drive, Baltimore, MD 21218, USA
[2]Department of Physics and Astronomy and PITT PACC, University of Pittsburgh, 3941 OHara Street, Pittsburgh, PA 15260, USA
[3]Department of Astronomy, Box 351580, University of Washington, Seattle, WA 98195, USA
[4]Department of Physics and Astronomy, University of Utah, 115 S 1400 E, Salt Lake City, UT 84112, USA
[5]Centro de Astronomía (CITEVA), Universidad de Antofagasta, Avenida Angamos 601 Antofagasta, Chile
[6]Chinese Academy of Sciences South America Center for Astronomy, China-Chile Joint Center for Astronomy, Camino El Observatorio, 1515, Las Condes, Santiago, Chile
[7]Department of Physics, New York University, 726 Broadway, New York, NY 10003, USA
[8]University of California Observatories, University of California Santa Cruz, 1156 High Street, Santa Cruz, CA 95064, USA
[9]Department of Physics and Astronomy, Johns Hopkins University, 3400 N. Charles St., Baltimore, MD 21218, USA
[10]Department of Physics and Astronomy, Haverford College, 370 Lancaster Avenue, Haverford, Pennsylvania 19041, USA
[11]Institute of Cosmology & Gravitation, University of Portsmouth, Dennis Sciama Building, Portsmouth, PO1 3FX, UK
[12]School of Physics and Astronomy, University of St Andrews, North Haugh, St Andrews, KY16 9SS, UK
[13]Department of Physics and Astronomy, University of Alabama, Tuscaloosa, AL 35487, USA
[14]Department of Physics and Astronomy, University of Kentucky, 505 Rose St., Lexington, KY 40506-0057, USA

ABSTRACT

The Mapping Nearby Galaxies at Apache Point Observatory (MaNGA) survey, one of three core programs of the fourth-generation Sloan Digital Sky Survey (SDSS-IV), is producing a massive, high-dimensional integral field spectroscopic data set. However, leveraging the MaNGA data set to address key questions about galaxy formation presents serious data-related challenges due to the combination of its spatially inter-connected measurements and sheer volume. For each galaxy, the MaNGA pipelines produce relatively large data files to preserve the spatial correlations of the spectra and measurements, but this comes at the expense of storing the data set in a coarsely-chunked manner. The coarse chunking and total volume of the data make it time-consuming to download and curate locally-stored data. Thus, accessing, querying, visually exploring, and performing statistical analyses across the whole data set at a fine-grained scale is extremely challenging using just FITS files. To overcome these challenges, we have developed `Marvin`: a toolkit consisting of a Python package, Application Programming Interface (API), and web application utilizing a remote database. `Marvin`'s robust and sustainable design minimizes maintenance, while facilitating user-contributed extensions such as high level analysis code. Finally, we are in the process of abstracting out `Marvin`'s core functionality into a separate product so that it can serve as a foundation for others to develop `Marvin`-like systems for new science applications.

1. INTRODUCTION

Large astronomy collaborations with dedicated facilities pursuing multi-year surveys are producing massive data sets at furious rates. The data sets from the current generation of surveys, such as the Sloan Digital Sky Survey (hereafter SDSS; York et al. 2000; Strauss et al. 2002), require more disk space than is available on personal computers and some moderate-sized institution-level servers. However, the next generation of surveys, such as the Large Synoptic Sky Survey (Ivezić et al. 2008) and the Square Kilometer Array (Braun et al. 2015), will create data sets that will be far too large for all but a few dedicated national-level facilities. The real power of these immense data sets comes from simultaneously leveraging multiple sources of information (e.g., at

bcherinka@stsci.edu

different wavelengths) about each object, so connecting the relevant data sources for a comprehensive analysis is critical. Since individual users cannot store the data locally and need to access portions of the data remotely, bandwidth is often the primary bottleneck. Speed increases in Internet bandwidth have lagged behind those in computer processors (i.e., Moore's law; Moore 1965) by 10% (Nielsen 1998); the effect of this lag has compounded over decades, up to the present, to exacerbate the gap. Consequently, only a subset of the data can be transferred. However, selecting this subset often requires access to the whole data set, which requires remote operations, especially queries.

SDSS was one of the earliest and remains one of the strongest driving forces in astronomy pushing the philosophy of public data releases that make astronomy a leader in open science. Crucially, these data releases are served with robust data distribution systems and come thoroughly documented. These two often-overlooked aspects have lowered the entry barrier and enabled thousands of professional astronomers and many times more public users to take advantage of this powerful data set. `Marvin` extends this mission by providing code to facilitate data use by professional astronomers, scientists in other fields (e.g., physics, computer science, and statistics), data scientists, citizen scientists, educators, and students.

The current phase (2014–2020) of SDSS, SDSS-IV (Blanton et al. 2017), consists of three simultaneous surveys, including the Mapping Nearby Galaxies at Apache Point Observatory (MaNGA; Bundy et al. 2015) survey. Legacy SDSS (York et al. 2000) took spectra of only the central regions of galaxies (Strauss et al. 2002), whereas MaNGA takes hundreds of spectra per galaxy arranged in a hexagonal grid across the face of the galaxy (Drory et al. 2015), using the SDSS/BOSS spectrographs (Smee et al. 2013) on the SDSS telescope (Gunn et al. 2006). Typically, there are 3 dithered sets of 3 individual exposures offset from each other which are combined into a data cube (Law et al. 2016; Yan et al. 2016; Yan et al. 2016). Thus, each object is not represented by just a single central spectrum, but rather a well-sampled grid of spectra.

Figure 1 illustrates the format of the MaNGA dataset. Each data cube consists of two spatial dimensions and one wavelength dimension. The one-dimensional spectrum at each spatial location can be interpreted in terms of measurements and physical parameters, yielding over 150 two-dimensional maps for each galaxy (Westfall et al. in prep.), including: gas emission lines, stellar absorption features, stellar surface density, star formation rate surface density, stellar velocities, and gas velocities. These maps can then be interpreted in terms of global properties of each galaxy: its mass in stars, its mass in dark matter, its total star formation rate, and other quantities. `Marvin` and the MaNGA maps for 4824 galaxies will be publicly released as part of Data Release 15 (Aguado et al. 2018).

In addition to its complexity, MaNGA's data volume is significant. MaNGA will observe over 10,000 galaxies (Law et al. 2015; Wake et al. 2017), more than an order of magnitude larger than previous IFU surveys, such as the Atlas^{3D} (Cappellari et al. 2011), DiskMass (Bershady et al. 2010), and CALIFA (Calar Alto Large Integral Field Area; Sánchez et al. 2012) surveys. All told, the final MaNGA data release will be 10 terabytes or about 1 gigabyte per galaxy in final summary data products, containing: data cubes and row-stacked spectra in log and linear wavelength sampling, derived analysis maps, and model template data cubes. Individual data releases contain multiple analyses of each galaxy, each optimized for different science goals, resulting in multiple versions, e.g different binning schemes, of the data cube and maps. The total volume for all of the MaNGA public data releases will be 35 terabytes due to re-analyses of the same galaxies as the data pipelines improve. Because of these re-analyses, if a given scientific paper is to be replicable, easy access to previous data releases must also be provided.

Further complicating analysis of MaNGA data is its coarsely-chunked storage across separate files for the spectra and derived property maps for each galaxy. Traditionally data are stored this way to optimize for an object-by-object catalog of files. This coarsely-chunked data makes querying on MaNGA's spatially-resolved data quite difficult without extensive manual preparation of all files and tracking of correct cross-matches, so queries can only easily be done on global properties. Exploratory analysis and visualization are cumbersome with coarsely-chunked data, which is compounded by the disconnected packaging of the spectra and maps. Finally, coarsely-chunked data unnecessarily strains bandwidth and disk space resources because superfluous data need to be transferred and stored. These challenges encourage traditional object-by-object analyses instead of innovative ones that leverage the statistically significant sample size of MaNGA.

This paper presents software to address these challenges. Section 2 describes the initial prototype and its inherent limitations, the core design philosophy of `Marvin` and the components involved. Section 3 describes the variety of client-based programmatic tools available in `Marvin`. Section 4 describes the front-facing web portion of `Marvin` which serves as the exploratory portal of entry for new users. The server-side features and back-end capabilities are discussed in Section 5. Section 6 describes a typical science use case for MaNGA and how `Marvin` streamlines its implementation. In Sec-

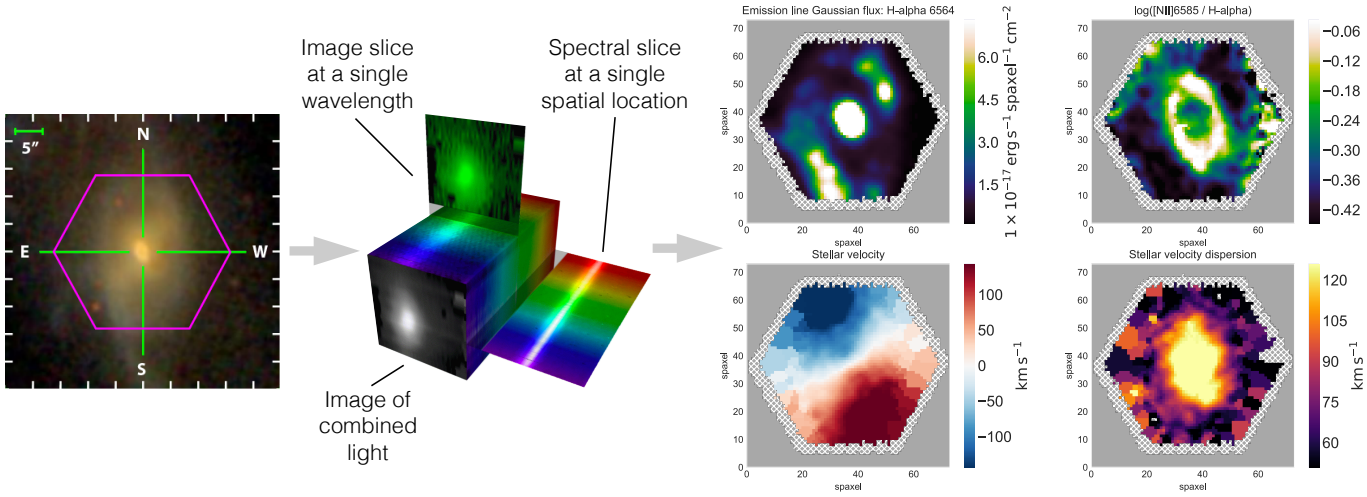


Figure 1. *Left*: *gri* image of MaNGA 1-596678 with the IFU field of view shown in purple. *Middle*: IFU observations produce three-dimensional datacubes, with two spatial and one spectral dimensions (credit: Stephen Todd and Douglas Pierce-Price; IFS Wiki; http://ifs.wikidot.com). *Right*: Spectral analysis of individual spaxels produces hundreds of two-dimensional maps for each galaxy spanning a wide range of physical properties. The four example maps shown for 1-596678 are Hα flux (top left), log([N II] λ6585 / Hα) flux (top right), stellar velocity (bottom left), and stellar velocity dispersion corrected for instrumental broadening (bottom right). The map plots were made using the `Marvin` code provided in the A.2.

tion 7, we discuss our current implementation strategy for engaging the long-term sustainability of `Marvin`. We summarize and discuss the future potential of `Marvin` in Section 8. Finally, a series of example code and tutorials are provided in the Appendix.

2. CORE DESIGN

2.1. *The `Marvin` Prototype*

To address the challenge of visually exploring MaNGA data, we developed a prototype version of `Marvin` that existed as a pure web-application. The prototype displayed optical images, spectra, and property maps for individual galaxies. These visual displays, in conjunction with a basic annotation system, proved useful for quality assessment of an early version of the MaNGA pipelines. The prototype also featured a simple query system and provided links to download the FITS data files.

The design choices for the prototype enabled rapid development, but ultimately limited its utility and sustainability. The images, spectra, and maps were static PNG files, which could not provide the interactive experience required for a complete visual exploration of the complex suite of available parameters. Queries could only be performed on global properties not local (spatially-resolved) ones. Data could only be accessed via large files that contained all of the spectra or property maps for a galaxy, making it impossible to retrieve just the spectrum or a single property of an individual spaxel. Because expanding the feature set of the prototype required creating new static files, the prototype was difficult to extend and time-consuming to maintain.

Furthermore, none of the components in the prototype web-application were usable in a command line form. Users were forced to reinvent the same visual and search tools if they wanted to use them programmatically. Such tools could serve as the basis for and be related to advanced programmatic analysis tools. Every user would end up developing similar tools but within different frameworks, such that each individual's analysis code would not be interoperable with that of other users.

These limitations of the prototype design failed to address any of the inherent challenges of the MaNGA data set. Thus, a complete redesign and refactor was required to fix these shortcomings, which led to a new design philosophy of `Marvin`.

2.2. *Design Philosophy and Core Components*

`Marvin`'s design philosophy focuses on eliminating the overhead costs and limitations of accessing the large, coarsely-chunked, and incompletely-linked MaNGA data set. Solving these issues enables on-demand data access, interactive visual exploration, minimal downloads, spatially-resolved queries, and statistical analyses at a spaxel-level. `Marvin` provides a feature-rich framework that serves as the building blocks for user-developed analysis tools that can be contributed back into `Marvin` to maximize code reuse and accelerate scientific progress.

`Marvin` is a complete toolkit designed for overcoming the challenges of searching, accessing, and visualizing the MaNGA data. The core design is centered around a few main components:

- A Multi-Modal Access (MMA) system that handles all data flow paths.
- An Application Programming Interface (API) based on the Representational State Transfer (REST) architectural style that handles all communication between the client and server.
- A `Brain`, a common core package that handles generic functionalities and abstracts common methods needed during data gathering.
- A programmatic `DataModel`, that simplifies handling of a large suite of parameters that may differ between data releases and formats.

`Marvin` combines and builds on top of these core pieces to provide the following additional tools:

- A suite of interconnected Python tools, all based off a core Python tool with the MMA system built-in, with two main tool types:
  - Data Product Tool: wraps your data products and retrieves specific chunks of data. (e.g., `Cube` or `Maps` in §3.1)
  - Query Tool: performs SQL queries against the remote data, with a pseudo-natural language syntax parser to simplify the user input.
- A Python `Interaction` class providing a uniform interface to the API, integrated into all the Tools.
- A web application, built on top of the Tools, for quick data visualization and exploration.

These tools work with each other, allowing for multiple entry points into the data, making it easy for users of various domain expertise (i.e from students to power-users), to access the data using the same suite of tools.

2.3. *Multi-Modal Access*

In the case of MaNGA, the amount of data produced (the final data release will be of order 10 TB) sits on the boundary of what a user can store and analyze locally with normal computing resources. Future surveys (e.g., the Large Synoptic Survey Telescope) will produce data sets many orders of magnitude larger than MaNGA's, thus requiring the development of new ways to access data.

One of `Marvin`'s core design choices is that data access should be abstracted in a way that makes the origin of the data irrelevant to the final user. `Marvin` accomplishes this goal with a Multi-Modal Access system with a decision tree that defines what access mode to use and the code implementation that executes it. Below we describe the data access modes: opening local files, searching local databases, or making API calls to a remote web server. Each of these data formats carries a series of advantages and disadvantages, but `Marvin`'s MMA allows users to leverage the advantages while minimizing the disadvantages.

Files (e.g. FITS) provide portable data that can be heavily compressed, and they are the current standard for astronomical data distribution. However, data access can be slow (especially from compressed files), and the data are usually stored in a way that requires a degree of familiarity with the data model. Moreover, doing searches and cross-analyses between multiple targets usually demands accessing a large number of files and keeping a significant amount of data in memory.

Relational databases solve some of these problems by storing the whole data set in an optimized and well-indexed way, which enables running complex queries efficiently, and provides quicker data access in most situations. In this case, the main disadvantages are the large size of a monolithic database (comparable to downloading all of the uncompressed files that compose the data set) and the difficulty of learning how to access data, especially compared to access via files.

Finally, data can be stored in servers (either as files or in databases) and accessed remotely via an API call that returns only the subset of data requested in the call. APIs are convenient for the user since they obviate the need to download data files to a local computer and can be used to abstract the data model. Their main downsides are that the internet is required to access the data and that applications that require access to large amounts of data can be slow to run.

`Marvin` Tools (see Section 3) include implementations that allow loading data from files, from a database, or via a series of API calls. However, once the data has been loaded, the Tools behave the same and produce the same results regardless of the data origin.

Figure 2 shows the decision tree followed by each tool to decide from where to load data. If the MMA is being run in "local" mode and a target identifier is provided (a plate-IFU or mangaid, which define a unique observation or a single target, respectively; see Yan et al. 2016), the code checks if a database is available and, if so, loads the data using it. If a database cannot be found, the default path file corresponding to that identifier and data release (generated as described in Section 2.3.1) is used, if the file exists locally. Alternatively, a file path can be passed to the MMA, in which case that file will be used.

In "remote" mode, an API call is done to a remote server with the target identifier and the data release as inputs; the remote server uses the same MMA in "local" mode to access the necessary data from a database

containing the complete MaNGA data set and returns them.

The default mode for `Marvin` is "auto" mode, which tries to access the data in "local" mode first and will try in "remote" mode upon failure. This order prioritizes local over remote data access because the former is usually faster, while seamlessly transitioning to the latter if the data is not available locally. See Appendix A.1 for an illustration of accessing an object with the MMA under different inputs and data origins.

In principle, it would also be possible to set up a system with a complete MaNGA database and use `Marvin` to access it locally. While setting up such a system would be non-trivial from a technical standpoint, there are situations in which it could be advantageous (e.g., in the case of an institution that wants to provide a local mirror of the MaNGA data set).

Figure 3 shows a high level overview of the user interface in `Marvin`. The user has two main access points: the local `Marvin` client or the web browser interface. While the browser interface communicates directly with the `Marvin` server, the MMA operating on the client-side decides whether to access data locally or remotely via API calls to the `Marvin` server. The `Marvin` server (following the MMA decision tree) first attempts to access data from a local database and will fall back to files when needed.

2.3.1. *Abstract Path Generation*

A machine-aware approach to file locations requires generalizing the ability to generate full paths to these files and removing all traces of the base filesystem root directory. In this way, `Marvin` can be agnostic to whether it is installed on a user's laptop or an SDSS host server. This layer of functionality is provided by the publicly available `sdss-access` (Cherinka et al. 2018) and `sdss-tree` (Cherinka & Brownstein 2018) software packages. `sdss-tree` provides the local system environment variable setup, allowing tools to understand the relative locations of data, while `sdss-access` provides a convenient way of navigating local and remote file paths. Paths to files are defined in a template format, specified with a shortcut name, plus a series of keyword arguments that specify variables within the filenames. This enables users to specify a robust path to any file simply by adjusting the input variable parameters. These packages are designed around relative path definitions, allowing a user to replicate a full environment by changing the definition of the base path. With a single root environment variable set by the user, these packages automatically create a local filesystem structure that mimics the filesystem of the SDSS Science Archive Server hosted at the University of Utah on which the full MaNGA data archive is stored.

For a given file, `sdss-access` has the ability to look up the full system path, generate the corresponding HTTP URL, and generate a remote access path for use with `rsync`. This flexibility allows `Marvin` to know precisely where to look for a given file locally and also quickly switch to a remote host when needed. `sdss-access` has the ability to download files from an SDSS server using multi-stream `rsync`, a technology derived from the SDSS Transfer Product (Weaver et al. 2015). This enables fast and robust file transfers, which are particularly helpful for speeding up downloads of many files. The hierarchy of files is created identically at the destination. As paths are added to the service, `sdss-access` eliminates redundant downloading by first checking for the existence of the file locally and only downloads files that do not currently exist.

2.4. *Marvin's Brain*

`Marvin`'s `Brain` is a core product that `Marvin` relies on and contains the management and overhead needed for regular tasks. There are many skills, tasks, and functionalities that have become more common, and are often required, to interact with modern astronomical data interfaces. Examples include items such as constant management of local paths to data files, learning how to write HTTP requests for accessing data served remotely, learning SQL to access data from databases, or even learning how to write web applications to serve data to others. These kinds of tasks often end up as logistical overheads that can be frustrating for end users, as they take repeated time to learn or implement and become barriers. These barriers can impede users' ability to do their science, which, at best, delays scientific discovery and, at worst, prevents accessing the necessary data altogether.

The primary design goal with the development of `Marvin` was to abstract away these overheads, and provide a framework that automatically handles much of this management. While `Marvin` is software specific to the MaNGA data set, many of these overheads are often independent of the type of data being served or accessed. To facilitate easier access and potential reusability of these features for other projects, we have placed these kinds of features into an additional core product, called the `Brain`, which `Marvin` depends on. Figure 4 shows the relationship between `Marvin` and its `Brain`. `Marvin`'s `Brain` (shown in blue) exists as base classes that sit underneath all of the components within `Marvin`. These classes act as templates that can be reused and customized for different applications. Our aim is to continue to migrate existing common `Marvin` features into the `Brain` so others can utilize the same tools.

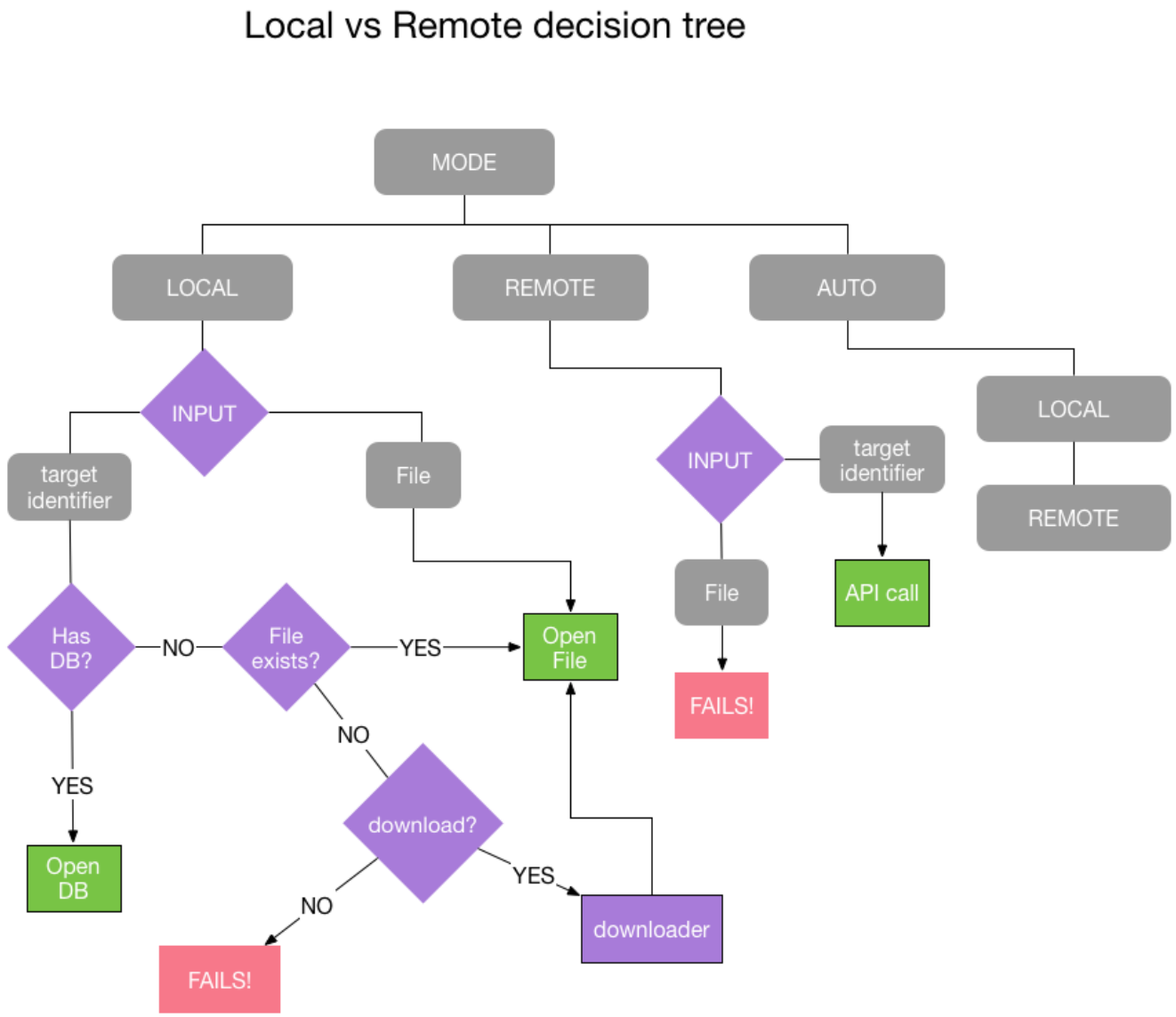


Figure 2. The decision tree for the Multi-Modal Access System. The Multi-Modal Access system operates in three possible modes: local, remote, and auto. See text for a detailed explanation.

2.5. *DataModel*

`Marvin` programmatically implements the unique MaNGA data model for each data release to abstract the Data Products for the MMA system and users. The MMA system relies on the `DataModel` to produce the same Data Product (e.g., `Cube` or `Maps`) from the correct data release regardless of whether it was instantiated from a FITS file, a database, or via the API. This abstraction makes scientific reproducibility much easier. It also enables users to programmatically navigate the Data Products without having to refer to the documentation. `Marvin` simplifies the data model for users by utilizing `FuzzyWuzzy`, a fuzzy string matching algorithm, to fix incorrect but unambiguous user input (e.g., "gflux ha" maps to "emline_gflux_ha_6564"). The `DataModel` is available as a standalone navigable object allowing access to the content and format of all MaNGA deliverables from a single location. Additionally, individual data models are attached to every relevant `Marvin` Tool, providing an internal lookup that all Tools use for self-consistency, making them robust against any changes to the underlying data files. As the format of the FITS files changes periodically between data releases, the structure of the Tools remains the same as the data model provides that intermediate go-between. Finally, the documentation for the data model is automatically generated (see Section 7.3) for reference.

3. PROGRAMMATIC TOOLS

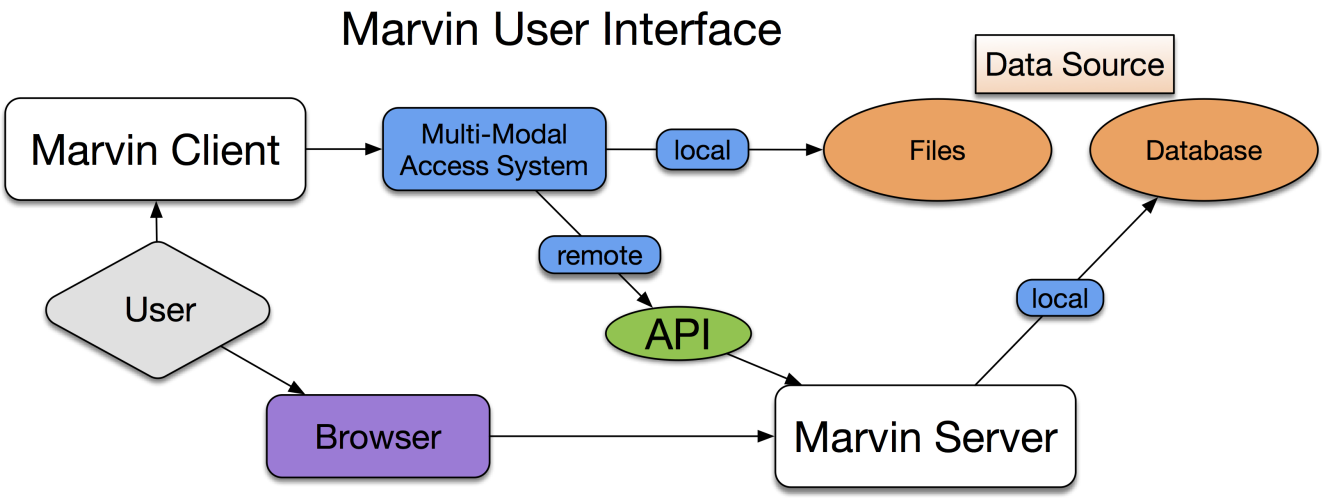


Figure 3. A high level user interface of `Marvin` and the Multi-Model Access system, depicting the two major paths of user flow through the system. Namely, via the browser which communicates directly to the `Marvin` server or via the `Marvin` client, which uses the MMA to decide on local or remote access.

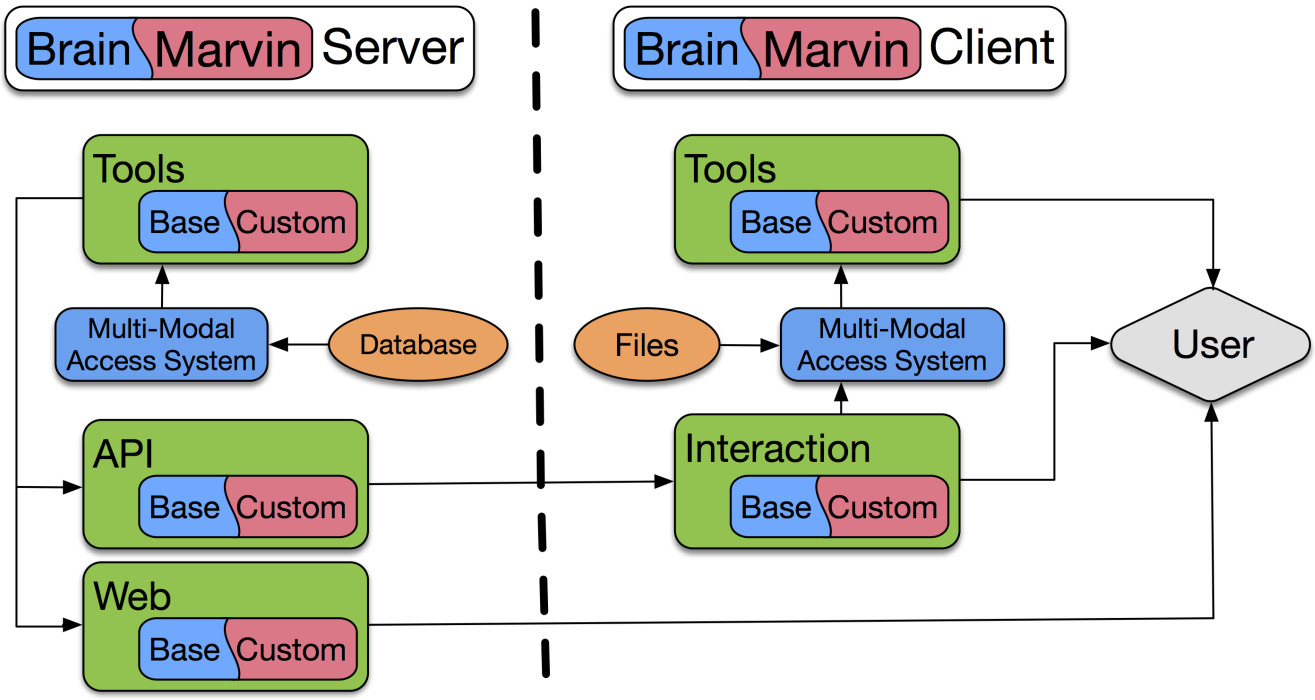


Figure 4. `Marvin` data flow and the relationship between `Marvin` and the `Brain`. Base classes in the `Brain` are subclassed into customized `Marvin` classes. All data flows via the Multi-Modal Access system. Client-side data flows through the Tools, and server-side data flows through the API or Web (through the Tools, `Interaction` class, or the browser).

`Marvin` provides a programmatic interaction with the MaNGA data to enable rigorous and repeatable science-grade analyses beyond simply visualizing the data. These tools come in the form of a Python package that provides convenience classes and functions that simplify the processes of searching, accessing, downloading, and interacting with MaNGA data, selecting a sample, running user-defined analysis code, and producing publication quality figures. `Marvin` Tools are separated into two main categories: Data Product Tools and Query Tools. The Data Product Tools are object-based and are constructed around classes that correspond to different levels of MaNGA data organization. The Query Tools are search-based and are designed to provide the user the ability to remotely query the MaNGA galaxy data set and retrieve only the data they want. `Marvin` also provides a built-in data model, which describes the science deliverables for every data release of `Marvin`. Overall, these tools allow for easier access to the data without knowing much about the data model, by seamlessly connecting all the MaNGA data products, eliminating the need to micromanage a multitude of files. Figure 5 shows a visual guide to all our tools, and highlights the interconnectivity between them.

3.1. *Galaxy Tools*

These tools cover four main classes, `Cube`, `RSS`, `Maps`, and `ModelCube`, that are associated with the analogous Data Reduction Pipeline (DRP; Law et al. 2016) and Data Analysis Pipeline (DAP; Westfall et al. in prep.) data products—namely, multi-dimensional data cubes, row-stacked spectra, derived analysis maps, and model cubes. The four main tools all inherit from a common core object, thus sharing much of their functionality and logic, such as the MMA. These tools are designed to do more than simply wrap and serve the underlying data and metadata contained in FITS files. Their goal is to streamline the users' interaction with that data and simplify common but often non-trivial tasks associated with handling the data. Via these tools, all data is delivered as `Astropy Quantity`s, with attached variance, mask, and any associated available properties. With `Quantity` variance and mask tracking, this enables robust and consistent arithmetic between any of the DAP `Maps`. Each tool has a built-in data model describing the format and content of the data it delivers. This data model also provides convenient top-level access to all properties available, with autocomplete navigation. Any given tool has convenient access to associated data products, as well as easy download capability for any data accessed remotely.

Features or functionalities that are common to multiple tools are designed as Python Mixin objects. These objects are designed as isolated pieces of code that can be "mixed in" with any other tool, giving that tool access to its parameters. Access to the NASA-Sloan Atlas (NSA) catalog (Blanton et al. 2011)[1] and the DAP summary file for instance are implemented in this manner. Extracting spaxels within a specified aperture is a common functionality delivered to all tools as a Mixin.

There are additional tools that are not associated with a particular MaNGA data file but instead map to objects related to the MaNGA data. These tools behave in much the same way as the core tools. They utilize the MMA, allow for remote file downloading, and are seamlessly integrated with each other. The `Plate` tool corresponds to an observed SDSS plate used during MaNGA observations. This object provides a list of all of the `Cube`s observed on a given plate, along with additional metadata associated with the plate, e.g., exposure numbers, observation date, etc. The `Image` tool provides interaction with the MaNGA IFU image cutout from SDSS multi-band imaging. It allows for quick display of the IFU image, over-plotting of the IFU hexagon, over-plotting of the individual IFU bundle or sky fibers, or generating an entirely new image at a custom pixel scale. Additionally a list of `Image` objects can be quickly generated and downloaded to the local client system. Image utilities also exist to quickly download a list of images in bulk using the streaming capability of `sdss-access`.

3.2. *Sub-Region Tools*

Marvin provides sub-region galaxy tools, which are designed to access individual components within the main MaNGA data products. `RSSFiber`, `Spaxel`, and `Bin` provide access to the row-stacked spectra from individual fibers, datacube spaxels, or bins (for binned DAP data), respectively. These tools come with convenient plotting functions, as well as access to all the DRP and DAP properties associated with a given element. The DAP produces data products with different spectral binning schemes for different science cases: unbinned spectra (SPX), spectra binned to S/N$\sim$10 using the Voronoi binning algorithm (VOR10), and a hybrid binning scheme (HYB10), with spectra binned to S/N$\sim$10 for the stellar kinematics, but emission-line measurements are performed on the individual spaxels. The "HYB10" binning type for DAP products has complicated the underlying binning scheme of spaxels. The `Spaxel` and `Bin` tools make the binning much more straightforward. Each `Spaxel` property contains information about whether it is binned or not, hooking into the `Bin` tool when appropriate. The `Bin` tool displays only the relevant information for the underlying prop-

[1] https://www.sdss.org/dr15/manga/manga-target-selection/nsa/

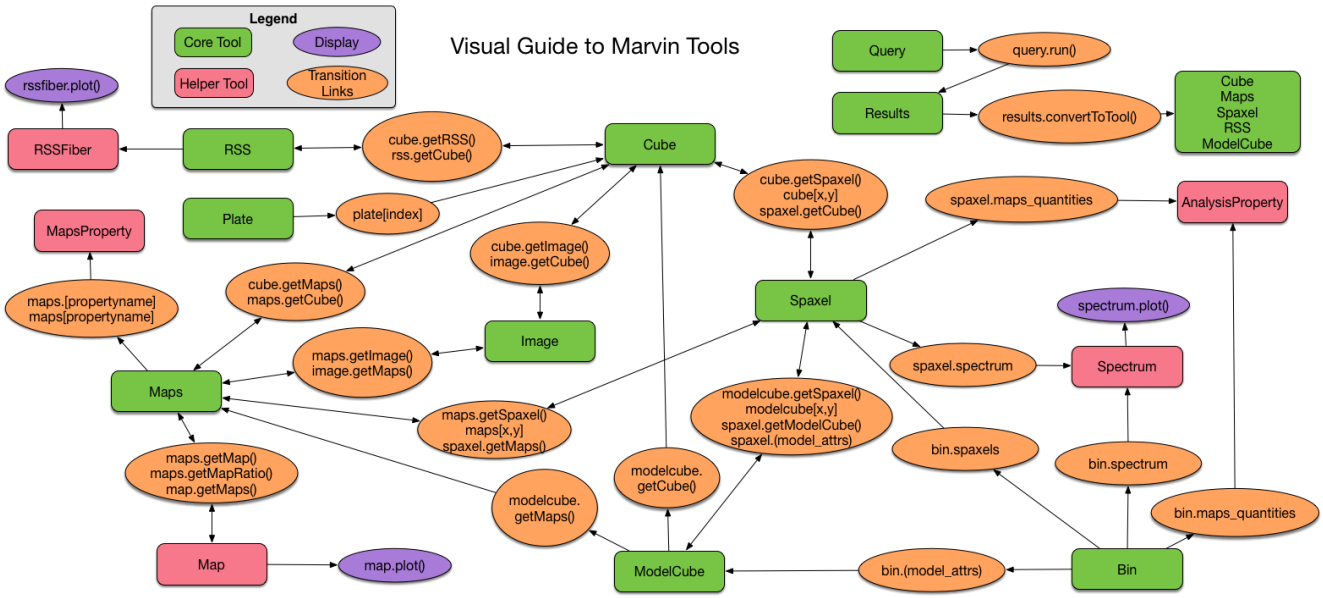


Figure 5. A visual guide of the programmatic Tools, highlighting the complex interconnectivity between the tools. Green icons represent core tool classes, with orange ovals showing the connections between them. Pink icons are helper tools, and purple icons are endpoints for visually displaying data.

erty and binning type, clearing up most of the obfuscation with accessing the "HYB10" binned files directly. From `Bin`, one can access all spaxels belonging to that bin, as well as generate masks for that bin.

3.3. *Query Tools*

`Marvin` provides tools for searching the MaNGA data set through an SQL-like interface, either via a web-form or a Python class. The `Marvin Query` system uses a simplified SQL syntax (see Section 3.3.1) that focuses only on a filter condition using boolean logic operators and a list of parameters to return. Not only does this simplify the syntax, but it automatically performs the incredibly complex table joins required to extract data from the MaNGA database. Users can query the MaNGA sample on global galaxy properties, similar to searching through the DRP and DAP summary files. In the near future, users will be able to perform intra-galaxy queries on individual spaxel measurements—a task that requires a database or loading all of the MaNGA spaxel data into RAM. Tutorials for querying with `Marvin` are available in the online documentation[2].

3.3.1. *Pseudo-natural Language Syntax*

Figure 6 shows an example of a MaNGA query in (1) natural language, (2) full SQL syntax, and (3) simplified pseudo-natural language syntax. While the query is relatively easy to describe in natural language, the full SQL syntax (red panel in Figure 6) is immensely complicated to construct, even if the user already knows how to write SQL queries. SQL queries consist of three main parts: a `select` clause, a `join` clause, and a `where` clause. Constructing the `select` and `join` clauses require detailed knowledge of the MaNGA database schema, table design, available columns, and the keys needed to join the tables. With the `Marvin Query` tool, rather than submitting the full SQL query, the user submits only a simplified `where` clause and an optional list of properties to return. The remainder of the query (the `select` and `join` clauses) is built dynamically behind the scenes, converted to raw SQL, and then submitted to the database. This allows the user to focus on the properties and their values in the selection criteria.

`Marvin` uses SQLAlchemy to map Python "model" classes onto each of our database tables and columns. This provides the base ability to dynamically build and submit SQL queries in Python. With these model classes, `Marvin` constructs a singular look-up dictionary containing a mapping between a string parameter name, in the form of `schema.table.column_name`, and its Python counterpart. This provides an automatic way of looking up the database location for a given parameter name, effectively removing the `select` clause. `Marvin` uses `networkx` to map those model classes onto a network tree, which allows the construction of a proper SQL `join` clause given any two input parameters across all tables in all schema in the database. Finally, `Marvin` uses a customized version of `sqlalchemy-boolean-search` to simplify the `where` clause to a simple input string.

[2] https://sdss-marvin.readthedocs.io/en/stable/query.html

This is a boolean parser which takes a string boolean filter condition, parses it, and converts to the proper SQLAlchemy filter object. The green panel in Figure 6 shows the pseudo-natural language equivalent of the desired query.

3.4. *Utilities*

Maskbit: MaNGA uses masks as a compact way to simultaneously convey information about the status of an object under many boolean conditions. The MaNGA pipelines produce quality masks at each processing stage, which allow users to filter out specific types of undesirable data when performing science analyses. During target selection, MaNGA likewise creates targeting masks that encode the sample or program under which an object was selected to be targeted.

The Maskbit class is a general purpose utility used by other Data Product Tools. It automatically loads the schema for a mask, which can be easily displayed for the user. It can then convert between the native integer value (e.g., 1025) to the list of bits set (e.g., [0, 10]) to the corresponding list of labels (e.g., ["NOCOV", "DONOTUSE"] for the `MANGA_DRP3PIXMASK` mask, which indicates that the spaxel has no coverage in the cube and should not be used for science). Users can create a mask by providing a list of labels instead of filtering bits. This class also enables searching on bits, which is particularly useful for target selection using the targeting masks.

Plotting Utilities: `Marvin`'s plotting utilities enable users to quickly display images, spectra, and maps of individual MaNGA galaxies or galaxy sub-regions. The plotting utilities also can put galaxies or sub-regions in context via scatter and histogram plots of query results. Beyond the image utilities, which have been described previously, and the spectrum plotting, which is a straightforward line plot, we will describe the map, scatter, and histogram plotting in more detail below. As a general philosophy, `Marvin`'s plotting utilities are designed to have smart defaults for quickly making useful visualizations while allowing for significant customization via standard `Matplotlib` methods.

One of the most difficult aspects of generalized map plotting is automatically setting the range of the color bar without being overly sensitive to poor measurements or outlier values. Map plotting automatically masks spaxels with poor measurements as flagged by the DAP or due to low signal-to-noise ratios. Users can tailor the masking by specifying flags, creating new masks, or changing the minimum signal-to-noise ratio. These masked regions are differentiated from areas outside of the IFU footprint to distinguish between regions with poor measurements and regions without data. To handle good but outlying measurements, `Marvin`'s map plotting does percentile clipping by default but allows for sigma clipping or a user-defined range. There also is a logarithmic option to help display properties with large dynamic ranges. It automatically uses a symmetric color bar for velocity maps since there is a natural zero point. `Marvin` also has an option for creating discrete color maps to show, for instance, spatial regions whose nebular line ratios are consistent with photoionization via star formation or a central AGN.

Querying is one of `Marvin`'s most powerful features. Yet it is difficult, if not impossible, to discover trends in large tables of text produced from a query. To that end, `Marvin` includes utilities to make scatter and histogram plots of query results. Queries in `Marvin` can return results with anywhere from a few to millions of data points, so `Marvin`'s scatter plot changes the underlying display technique depending on the number of data points (see Figure 7). Fewer than 1,000 data points are shown individually (7a), 1,000–500,000 data points are shown as a hex binned density distribution (7b), and more than 500,000 data points are shown as a scatter density map (7c) that is responsive even with millions of data points. By default, scatter plots show marginal histograms with the mean and standard deviation. Users can also create histograms separately from a scatter plot and extract the data points in each bin.

Analysis Tools: At the time of publication, we have prioritized development of aspects of `Marvin` required for interfacing with the MaNGA data over providing downstream analysis tools. However, `Marvin` is ideally suited to serve as a foundation for analysis tools that extend its functionality to additional processing steps. One such analysis tool that has already been developed is a tool to classify different regions of a galaxy according to classical emission line ratios.

As discussed by, e.g. Baldwin et al. (1981) (BPT hereafter), the nebular permitted (Hα, Hβ) and forbidden emission line transitions (e.g., [O II] λ3727, [O III] λ5008, [N II] λ6585) are commonly strong and easy to detect in galaxies that contain significant quantities of gas. Since these transitions have different ionization potentials, their relative strengths encode information about both the metallicity of the gas and the hardness of the radiation field emitted by the source of ionizing photons. As such, easily-measured line ratios such as [O III]/Hβ and [N II]/Hα can be used to discriminate between H II regions produced by thermal (i.e., star formation) and non-thermal processes (e.g., shocks and active galactic nuclei).

`Marvin`'s BPT tool returns masks in which individual spaxels have been classified as "star-forming", "Seyfert" or "LINER-like" line ratios, such that a user can then plot diagnostic diagrams, categorial maps of the classifications, or maps filtered by these classifications (for instance, plotting the Hα flux for star-forming regions).

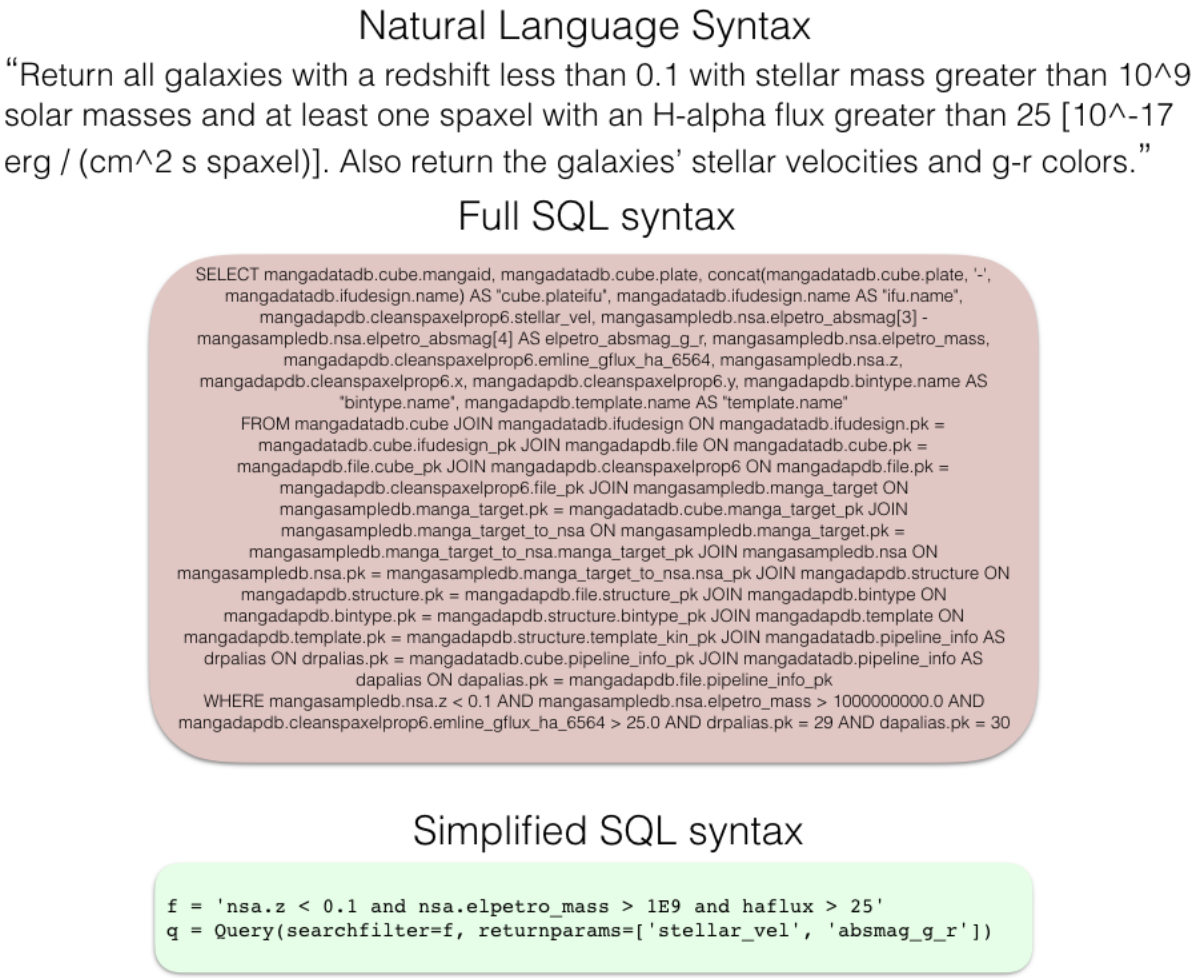


Figure 6. An example query on the MaNGA data set. The top describes the query in natural language syntax, i.e., how a user would describe it. The red panel shows the full SQL syntax needed to perform the same query on the MaNGA database. The green panel shows the corresponding pseudo-natural language syntax, and its use in `Marvin`.

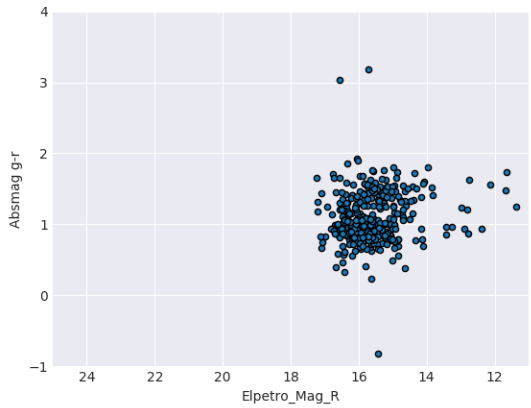
Such analyses have revealed significant clues as to the physical origins of the ionizing photons, indicating for example that in many sources observed to have LINER-like line ratios in SDSS single-fiber spectroscopy the gas is spatially extended and likely ionized by hot evolved stars rather than a central AGN (Belfiore et al. 2016).

3.5. *Contributed Code*

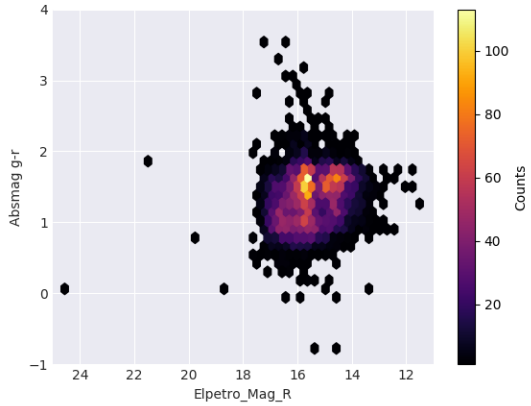
While the core of SDSS data releases centers around its base projects' science deliverables, smaller teams frequently contribute added value to its core deliverables with additional science products. These value-added data products or catalogs (VACS) are derived data products based on the core deliverables that are vetted, hosted, and released by SDSS in order to maximize the impact of SDSS data sets. To increase the visibility of MaNGA VACs, `Marvin` has hooks to allow users to contribute small pieces of code that plug their VACs into the overall system, immediately connecting their VAC into the larger suite of `Marvin` Tools and MaNGA Data Products. Each contributed code piece is well-documented, adheres to the overall standards set by SDSS, and contains the proper software credit for the user.

The core design principles of `Marvin` are to perform most of the legwork for the users, making access as easy as possible, while allowing users to contribute their own code to help expand `Marvin`'s functionalities. For VACs, contributors create a new class defining their VAC based on a pre-defined base class which provides unique target identifiers and automatic file retrieval methods needed to extract specific data from files. Contributors simply define the name of their VAC, the unique file path parameters, and a single method returning the data content. Contributors do not need to implement access to the core data products, which is already handled by `Marvin`.

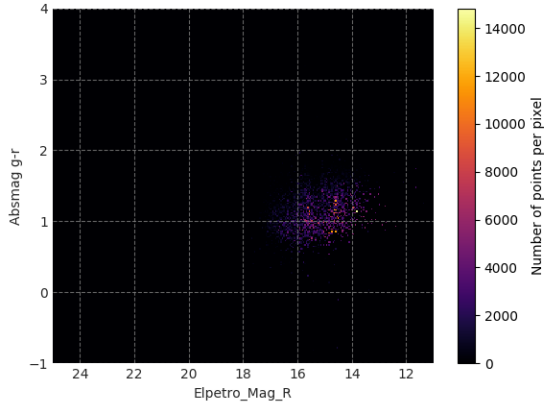
More generally, the `Marvin` code is structured to ease



(a) Matplotlib scatter plot for results with less than 1000 points.



(b) Matplotlib hexbin plot for results with between 1000-500,000 points.



(c) Matplotlib scatter density plot using https://github.com/astrofrog/mpl-scatter-density for results with more than 500,000 points.

Figure 7. Scatter plotting capability from the `Marvin Results` Tool. Depending on the number of results, `Marvin` plots using a straight scatter plot, a hex-binned density distribution, or a scatter density map. Note: this figure uses the same scatter plot in each subfigure to illustrate the differences in plotting styles rather than separate plots with more accurate result counts.

contributions of drop-in utility or analysis methods that add functionality to `Marvin`. These functions can manipulate or extract data from existing `Marvin` Tools, perform some analysis, or return a plot or data. The BPT tool from the previous section serves as an example of such a drop-in function that easily wraps the existing Tools. Users are encouraged to contribute `Marvin`-based analysis code back into the project so that others can take advantage of their efforts.

4. MARVIN WEB

The web, or browser-based information gathering, is often the first entry point for any user new to a field. Poor web design (e.g. cluttered content, complex interfaces) can quickly discourage users from interacting with the delivered content. `Marvin` provides a web frontend that aims to be as intuitive and streamlined as possible, with a focus on quick visual exploration of the MaNGA data set, leaving more rigorous analysis to the programmatic `Marvin` Tools or the user's own scripts. This minimal but interactive interface encourages users to quickly engage with MaNGA data and, when ready, seamlessly transition into more advanced environments. Our web component is built using `Flask`, a Python-based, lightweight, micro web-framework. `Flask` allows for quick deployment of a web application with minimal effort. It contains its own built-in web server for small scale deployment, or can easily be integrated into more advanced web-servers for production deployment. It has built-in hooks for modularity and extensibility, and employs a templating system for writing front-end code like HTML or Javascript in a modular way.

`Marvin` Web currently provides the following features:

- a Galaxy page, for detailed information and interaction with individual galaxies in MaNGA
- a Query page, for searching the MaNGA data set using the simplified SQL pseudo-natural language syntax described in Section 3.3.1
- a Plate page, containing all MaNGA galaxies observed on a given SDSS plate
- an Image Roulette page that randomly samples images of MaNGA galaxies, useful for browsing the wealth of variety in the 10,000 galaxy sample

The Galaxy page (see Figure 8) provides dynamic, interactive, point-and-click views of individual galaxies to explore the output from the MaNGA DRP and DAP, namely, spectra and map properties, along with galaxy information from the NSA catalog. In contrast to the prototype, this page is completely interactive, with more galaxy metadata. These interactive features are deployed using a variety of third-party Javascript libraries:

Dygraphs for the spectral viewer, Highcharts for the map and scatter plot viewers, OpenLayers for the interactive optical image display, and D3 for the box-and-whisker plots.

The Query page (Figure 9) provides the entry point for quickly searching through the MaNGA dataset. In contrast to the prototype, this page provides search capability for all properties in the DRP and DAP summary files, with minimal impact on the design interface of the page. The search capability will soon be extended to the entire suite of MaNGA parameters. It is built on top of the Marvin Query tool, providing a single simple interface, for both the web and tool, that one needs to become familiar with. In addition, the query system can be easily extended for both web and client users at the same time. Performing a query produces a navigable table of results, with each row linking to the individual galaxy. One can optionally switch to a postage stamp view of all galaxies returned in the query subset.

While the input structure to the Marvin Query tool is simplified greatly from the underlying full SQL statement, the syntax can still be complicated to learn. Some users may find it cumbersome, delivering confusion instead of intuition. The Query page also includes an interface for dynamically constructing a SQL statement in a guided manner (see Figure 10). This interface provides a series of parameter drop-downs which, in conjunction with operators and values, can be used to build conditions, and combined together with boolean operands. A web video tutorial[3] is available highlighting general usage, with more information available in the online documentation.

Because the web components are built on top of the Tools, all the galaxy and query features can be mapped to an underlying equivalent Marvin tool command. This allows users to recreate what they experience in the Web with the Marvin Tools locally on their system. On each page we provide feature-specific code snippets that indicate the equivalent commands for viewing galaxy maps or spaxels, or querying the dataset. These snippets can be copied and pasted directly into the local iPython terminal.

For the back-end, Flask provides the basis for the framework as well. It can be run in a "debug" mode for rapid development or be served in a production environment. For production deployment, Marvin is run using the NGINX web-server, with uWSGI acting as the gateway interface between the Flask web-app and NGINX. Flask provides the basic framework on which the Marvin API and the web-facing front-end is built. Our back-end Flask routes are built using the same suite of Marvin Tools available to the user on the client side. In this manner, we can build a single tool for the user while also using it to provide the same content directly over the API or integrated into the front-end as a web features. The server-side Marvin uses the same MMA system to determine data location, pulling first from a local database hosted at Utah, then from the files located on the Science Archive Server (SAS) filesystem.

5. MARVIN BACK-END

5.1. *REST-like API*

To provide remote data access, Marvin employs a REST-like web API, which defines a set of rules for remote data acquisition through HTTP request methods (i.e., GET and POST). The API handles all requests and responses between the user and server. There are three ways to interface with the Marvin API: directly through HTTP (low level), with a Python helper class (mid level), or via Marvin Tools (high level) (see Figure 11).

The lowest access level provides direct HTTP access to the API routes. Our API routes use the underlying Marvin Tools and provide remote access to the most commonly desired features of the MaNGA data set. A list of the available routes, and what data they provide, can be found in the online documentation[4].

While an experienced user can directly use the HTTP routes to retrieve data, not everyone is familiar with how to handle HTTP requests and responses. The middle layer wraps the direct API calls into a Python Interaction class. The Interaction class utilizes the requests package to handle GET/POST exception handling, set default request parameters, check the response, and provide convenience methods for parsing return data into Python data-types.

At the highest level, the Interaction class is built into the core Marvin Tools (and any Tools customized from them), providing remote access ability to all Tools. Marvin contains a lookup dictionary for resolving API URL shortcuts into their full route names. This dictionary allows each Tool to understand its required remote call and provides robustness against server-side API route changes. In this access level, the user takes a hands-off approach to API requests. The Tool determines when to access data locally or remotely. If remote data access is required, it performs the API request without user input and shapes the data properly upon receiving the response. When using the API, the Tools employ a lazy-loading approach to remote data to

[3] https://www.sdss.org/dr15/manga/manga-tutorials/marvin-tutorial/marvin-web/

[4] https://sdss-marvin.readthedocs.io/en/stable/api/web.html

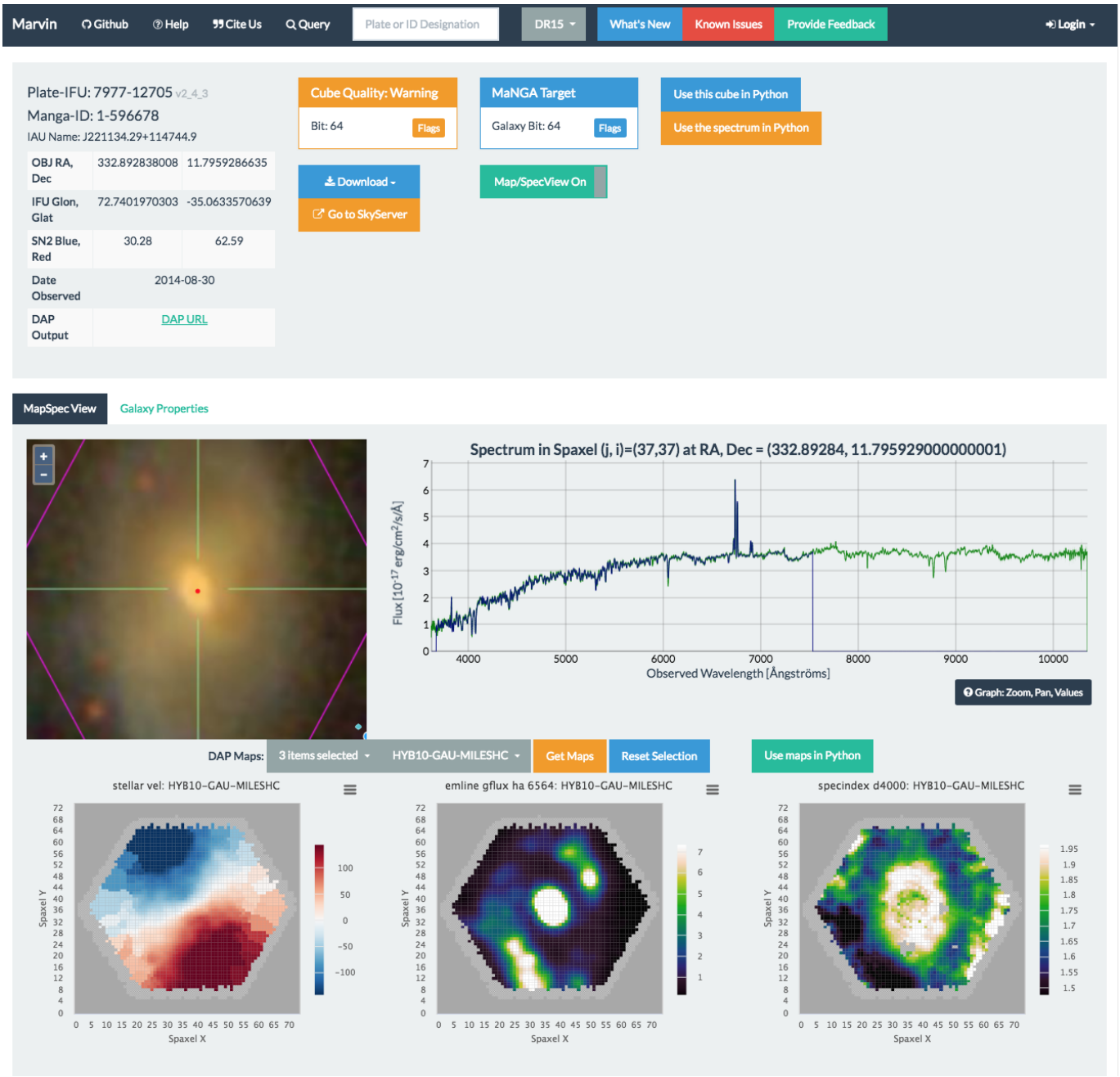


Figure 8. The `Marvin` web Galaxy page, highlighting the interactive point-and-click feature. Users can dynamically interact with individual galaxy spaxels and maps. Clicking anywhere within the optical image, or the DAP maps, retrieves the spectrum at that spaxel location. The interactive spectrum displays both the flux (green) and model fit (blue) for the selected spaxel. Marvin displays three maps by default: the Hα emission line flux, the stellar velocity and D4000 spectral index. Dropdown menus provide additional maps to display or maps of different binning schemes.

minimize server load. The API returns the minimum amount of information needed to satisfy the user's request. Additional information requested through the Tools is acquired through additional API calls.

5.2. *Database*

All MaNGA data is stored in a PostgreSQL relational database. The database is the bedrock data storage component of `Marvin`. It provides the basis for interactive visualization in the web, spatially-resolved queries, and selective data retrieval. For each release of MaNGA, we store the metadata and raw spectral output from the DRP and DAP pipelines. We currently store data from MaNGA internal data releases, referred to as MaNGA Product Launches, MPLs, 4-7, and the public data release DR15. The current size of the MaNGA database is 9 TB for 20153 galaxies across MPLs 4-7. The database contains three main schema: `mangadatadb` which con-

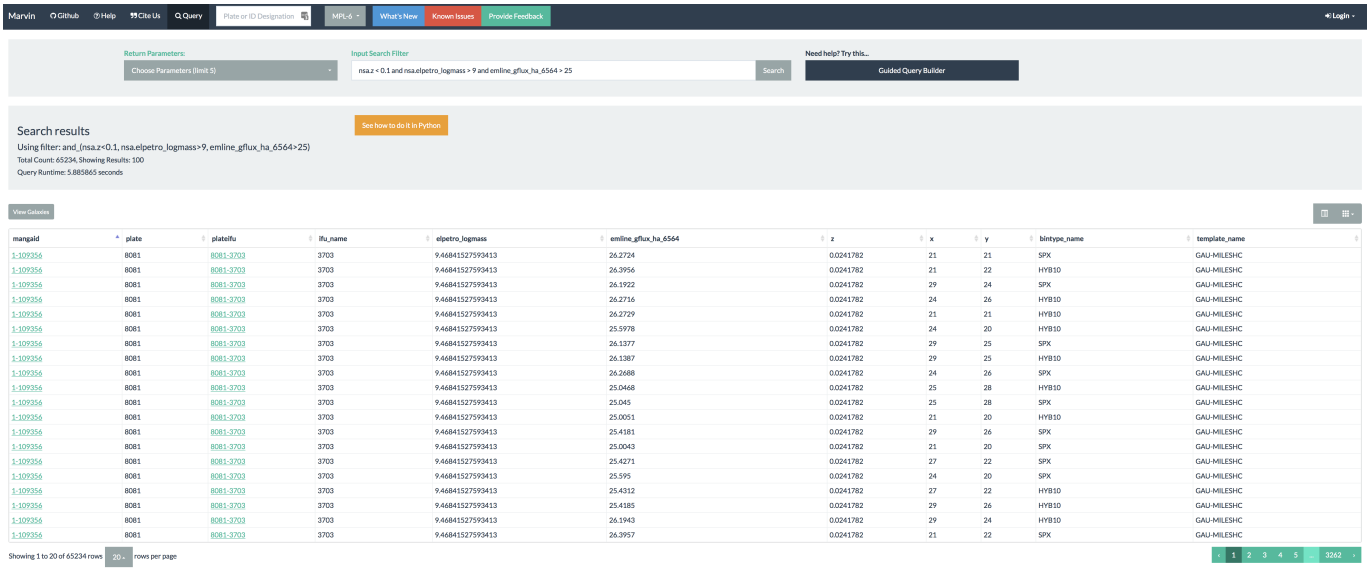


Figure 9. The **Marvin** web Query page, highlighting an example query using the simplified **Query** tool syntax, presents the results in a navigable table. The basic Query page components consist of a simple text box for inputting a search filter, and a drop-down menu of a subset of available parameters to return in the query. Additionally there is a button to construct a search filter in a guided fashion.

Figure 10. A guided SQL builder for constructing complex queries on the **Marvin** web Query page. This query maps to the search "return all galaxies with a redshift less than 0.1, a stellar mass greater than 10^9 solar masses, and at least one spaxel with Hα flux greater than 25 $10^{-17} erg/cm^2\ s\ spaxel$".

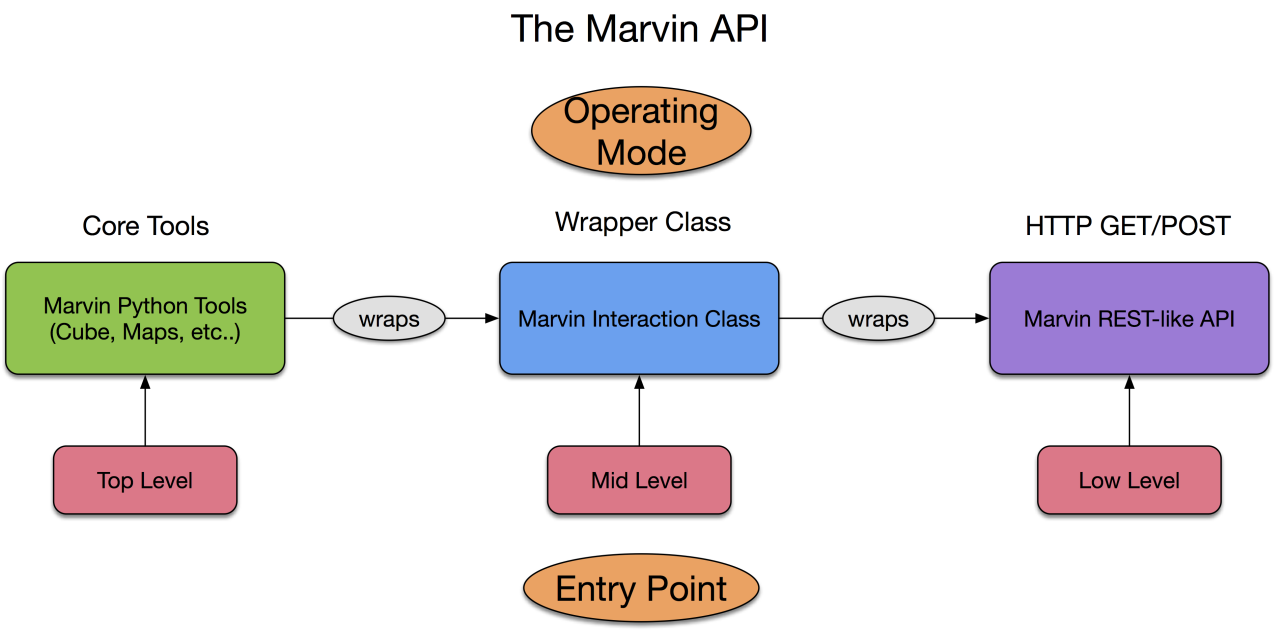


Figure 11. The three layers of the `Marvin` API highlighting the process of a remote Tool call. The top level (left) sits the `Marvin` Tools and is the main entry point for users. The middle layer (middle) provides a wrapper class for handling request logistics. The low layer (right) performs the standard HTTP requests. Users naturally start at the top-most layer, but may use any layer for performing remote requests.

tains the DRP output; `mangadapdb` which contains the DAP output; and `mangasampledb` which contains information on MaNGA targets and the NSA catalog. In addition, there is an auxiliary schema for miscellaneous data and a history schema which stores user and query metrics. The main schema designs can be found in the online documentation[5].

5.3. *SAS Filesystem*

The SAS filesystem is the data warehouse for SDSS. Hosted at the University of Utah and mirrored at the National Energy Research Scientific Computing Center, the SAS includes all of the raw and reduced SDSS data, including the intermediate and final data products from each survey's data reduction or analysis pipelines and value-added catalogs (∼1000 TB). The SAS serves both the collaboration (through a private gateway) and the public. The filesystem structure is organized hierarchically by survey to facilitate easy navigation. Both the software and data products are under version control, and the versions are explicitly included in the file paths. The explicit versioning in the file paths allows for consistent access and rapid deployment for internal and public data releases. All software and data products are frozen on a schedule set by the SDSS collaboration and tagged to maintain a self-consistent, reliable, and robust data system. These immutable tags and frozen reduction versions ensure the reproducibility of high-quality products. By developing `Marvin` to work on top of this structure, we can consistently deliver an archive-quality data product to the community, mitigating concerns about underlying intermittent data changes.

5.4. *Authentication and Access*

`Marvin` provides access to MaNGA data for both the SDSS collaboration and the public astronomy community. The SDSS collaboration provides data access rights for a proprietary period, so `Marvin` has collaboration-only and public access modes. The collaboration-only access mode provide access to the private gateway for both internal data releases and public data releases. In contrast, the public access mode provides access to the public gateway for only the public data releases. Approved SDSS collaboration members must authenticate with `Marvin` before access is granted to the private gateway. For the Web, `Marvin` uses the `Flask` extension `Flask-Login`, which uses session-based cookies to handle all login and authentication. For the API, `Marvin` uses `Flask-JWT-Extended`, which authenticates via JSON Web Tokens. After supplying their credentials in a Unix standard `netrc` file, the user is allowed to login and receive a valid token. The token is inserted into every API request and authenticated on the back-end.

[5] https://sdss-marvin.readthedocs.io/en/stable/api/db.html

6. WORKFLOW

Figure 12 highlights an example workflow going through the stages of Sample Selection, Data Access, Data Interaction, Data Linkage, and Interpretation. Prior to `Marvin`, the workflow for an analysis of MaNGA data, as indicated in the middle panel, would typically consist of a user selecting galaxies based on global galaxy parameters, downloading all the data files for those galaxies locally, then using existing tools (e.g., Astropy) to load the data into generic (i.e., not MaNGA-specific) FITS objects in a programming environment. To retrieve all relevant information for a target, the user must load, access, and construct the spatial links between data in separate files. Finally, users must write their own custom, often reinvented, analysis tools to visualize and interpret the data for their science.

The `Marvin` framework streamlines the existing workflow as shown in the rightmost panel of Figure 12. `Marvin` enhances existing workflow steps (shown as red text) and obviates workflow steps that require logistical overhead effort for data handling (shown in gray dashed boxes). While the methods involved in the existing workflow are functional, they contain the following problems, which `Marvin` redresses:

- The first four tasks in the workflow, i.e. selecting a target sample, downloading and linking FITS files, are easy to describe but require moderate effort to implement, which is compounded when iterating over selection criteria during exploratory analyses. In contrast, `Marvin` provides functionality to handle these ubiquitous tasks. In particular, `Marvin`'s front-end web interface enables rapid preliminary visual exploration without downloading data or writing code.

- The selection of galaxies to analyze can only be done on global quantities, not the maps nor the spectra. In contrast, search capabilities in `Marvin` are far more powerful, flexible, and detailed. `Marvin` can perform complex queries on the maps and spectra (e.g., search for galaxies with a high star formation rate surface density near the center).

- Unnecessary data is inevitably downloaded, since only entire files (containing the entire data cube or hundreds of maps) are available for download, increasing bandwidth and disk resources. In contrast, `Marvin` adds substructure to the data that downloads only the explicitly requested data (i.e.,

a single map or spectrum), minimizing bandwidth and local disk use, if desired.

- Individuals must build their own tools to manage the download of data to the local server, which can be complicated to manage efficiently without substantial effort. Additionally individuals must define a different set of tools for accessing remote data versus local files. These logistical issues pose a significant barrier to new users. In contrast, `Marvin` comes with such tools that automatically avoid multiple downloads of the same data. The same set of `Marvin` tools can be used in different hardware locations (i.e., with either local or remote access to the data) with only a single configuration change.

- To compare map quantities and spectra, individuals must build their own tools to link spatial locations in the maps to spectra. In contrast, most of the detailed data access tools are built into `Marvin`, and internally perform all necessary linkage in a standardized fashion.

- Visualizing the data is cumbersome and time-consuming as it requires all data be local, and relies on manual plotting scripts or the repetitive use of third-party tools. In contrast, `Marvin`'s web interface and Python package provide visualization tools for fast iteration and exploratory analyses.

- Individuals' analysis code remains siloed and is not reused. In contrast, `Marvin` includes some analysis code and serves as a foundation and repository for shared analysis code, which minimizes code duplication across researchers and projects.

`Marvin` is structured as a complete ecosystem such that the entire workflow can be performed in a single Python environment or program, but its modular design allows many aspects of `Marvin` to be used independently of each other. Data can be accessed either through `Marvin`'s provided Tools, or downloaded using `Marvin` but imported and analyzed with other tools. This flexibility makes `Marvin` a useful tool to a broad range of astronomers. See Appendix A.3 for an example `Marvin` workflow that examines the metallicities of star-forming spaxels in MaNGA galaxies.

7. SUSTAINABILITY

Sustainability is an important part of any software's longevity and usefulness within the community. Good software is well-documented, easy to maintain, and guided by productive interactions between its users and developers. Software development is often critically misunderstood to occur in an isolated environment and in static snapshots with little input from the community of users. Successful software adoption and development depends on easy installation, well-understood code, an open and eager community, quick identification of breaking changes through continuous testing, and rapid patch updates through continuous deployment. Below we highlight steps that we have taken to make `Marvin` more sustainable in the long term.

7.1. *Deployment*

To streamline installation and reach as wide an audience as possible, we tag and deploy versions of Marvin using the Python Package Index (PyPI)[6]. Packages on PyPI are installed with the `pip` package, which simplifies installation by handling the package dependencies. A `pip`-installable package simplifies user installation handling all software dependencies automatically. Installing software packages with a large number of dependencies can interfere with a users' local environment. To resolve this issue, we also provide full `conda` environment installations[7]. For users with the Anaconda distribution of Python, these environments will install `Marvin` in a self-contained virtual environment that does not affect a users' default environment.

7.2. *Open Source Code*

SDSS supports and encourages open source software development for all its projects and advocates for a fully transparent software cycle from development to release. We have adopted the BSD 3-Clause open source software license as it allows for the most complete freedom of use. The `Marvin` code and its SDSS dependencies are versioned using `git` and hosted in public Github repositories. To promote consistency among the SDSS software projects, SDSS encourages all members to adhere to a common set of coding standards and practices[8], which we have adopted in the development of `Marvin`. We additionally tag all versions of our software with a trackable DOI and host it on Zenodo[9].

7.3. *Documentation*

Software is only helpful if users understand how to use it, which means providing thorough and well-organized documentation. `Marvin`'s documentation is structured hierarchically so that it completely covers all of the public-facing code and yet remains easily searchable. The documentation also contains worked exam-

[6] https://pypi.org/

[7] https://anaconda.org/sdss/sdss-marvin

[8] https://sdss-python-template.readthedocs.io/en/stable/standards.html

[9] https://doi.org/10.5281/zenodo.596700

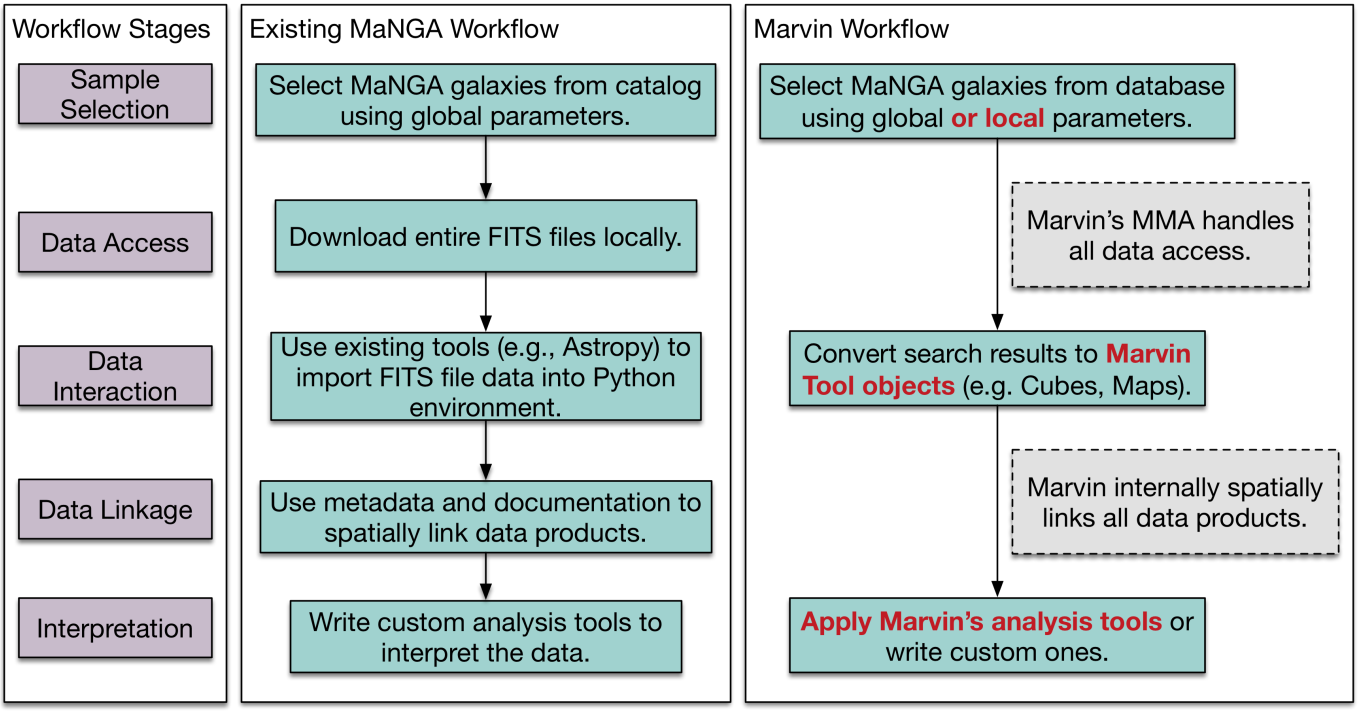


Figure 12. Typical MaNGA workflow stages from sample selection through interpretation (left column) performed with existing tools (middle column) and `Marvin` (right column). At each stage, `Marvin` either enhances the existing capabilities (highlighted in red) or automatically handles tasks (gray boxes).

ples that highlight commonly used workflows to associate scientifically-related functionality within the code base.

`Marvin`'s documentation starts with the docstrings in the code, which are included in all of the public-facing code. Users can access the docstrings on Github to understand the documentation in the context of the code or just simply and quickly view the docstrings while working in an interactive python terminal. The docstrings also serve as the input for the API reference created by the `Sphinx` package, which is the basis for web-based documentation pages.

`Marvin`'s web-based documentation[10] is hosted on Read the Docs, which automatically generates the documentation web pages (using `Sphinx`) every time the code in the "master" branch changes on Github. Using `Sphinx` with Read the Docs ensures that the documentation is consistent between the code base and the online documentation. This workflow also enables versioned documentation that corresponds to the versions (tags) of the code, including the data model. Versioned documentation is critical for understanding how the code base has changed over time and its potential effects on out-dated user code.

An important aspect of the usability of documentation is how quickly users can find answers to their questions. Read the Docs features a search bar that can be used to find keywords within all of the online documentation. We also have written documentation pages to help guide users through the code and provide paths for digging deeper into the documentation in useful directions. The documentation contains use-oriented pages for the major classes and other key aspects, such as installation. At a higher level, there are various tutorials on topics like first steps, querying, and plotting. For more complete science examples, there are Jupyter notebooks that are centered around addressing actual science questions. Finally, there is a cheat sheet that is a one-page quick reference guide to help users remember syntax and function names.

The documentation has benefited tremendously from user feedback within the SDSS community received during in-person workshops, in remote tutorial sessions, and via submitted Github Issues. Actual user feedback is invaluable, since the developers have a very different perspective that sometimes blinds them to gaps in the documentation.

7.4. *Error Logging*

To better track when errors occur as users utilize `Marvin` in their scientific workflows, all errors are cataloged using Sentry, an error-logging client. Rather than solely relying on users to report errors, Sentry is an open-source error tracking client that helps develop-

[10] https://sdss-marvin.readthedocs.io

ers monitor and fix crashes in real time. The Sentry client easily integrates into many existing frameworks, including Python and Javascript. Upon an error within `Marvin`, Sentry will catalog the error and full traceback along with the relevant software, user, and system information. Sentry allows us to track when and how often errors occur, automatically create Github Issues, and generally boost the efficiency at which `Marvin` failure modes are identified.

7.5. *Testing*

Testing is an integral part of developing and maintaining any software package. Simply put, tests confirm that code executes as intended; however, the benefits extend beyond the confidence inspired by passing tests. Testing speeds up development by enabling automated checking for regression bugs, where a code change unexpectedly breaks existing functionality, so developers immediately know when a breaking change is introduced. Testing strongly encourages other good software carpentry practices, such as writing short functions that only do one thing. Tests significantly lower the risk of refactoring code because they ensure consistent behavior of the code after refactor. Generally speaking, tests reduce the effort required to maintain a code base.

7.5.1. *Python Testing*

One of our goals is to have complete test coverage of the Python code since it underlies all aspects of `Marvin`. `Marvin` uses the `pytest` package because it requires less boilerplate code relative to the Python standard library package `unittest`; its multi-level fixtures; the ability to parametrize fixtures; its monkeypatching and mocking options to simulate environments or function calls; and its quick iteration and inspection of test failures. These features are particularly helpful for ensuring consistent behavior across data releases, analysis and binning types, and data access modes.

7.5.2. *Continuous Integration with Travis-CI*

Software teams are often developing in parallel with one another, and operating on multiple threads at the same time. This can lead to conflicts and lost time when merging threads back together or when modified or new code introduces breaking changes. Continuous integration allows a code base to test code changes in a continuous fashion in real time. `Marvin` uses the Travis-CI system for automated testing. Travis-CI is free for all projects hosted on Github. Once a project's repository is connected to Travis, any commit to the code triggers Travis to set up a virtual environment and run the full test suite. Travis can be configured to run the test suite under different environment conditions, such as different versions of the programming language.

7.5.3. *Code Coverage with Coveralls*

Code coverage is a method for determining how much of the code is covered by the tests. As the tests run, the lines of code touched by the tests are compared against the total number of lines, and a coverage report is prepared for every code file, and the software package as a whole. Coveralls is an online interface for visualizing and tracking the code coverage over time, designed to ensure the test coverage of the code is always increasing over time. It can be connected into existing CI services for real time updates. After Travis successfully completes a test run, the coverage output is pushed to Coveralls, where developers can see the progress over time, and examine individual files for lines of code untouched by tests.

7.5.4. *Framework Testing*

`Flask` comes with a test server designed to simulate running a real web server. It exposes the web and API routes so that tests can be properly written against them. All back-end web and API routes are fully tested for general compliance and for the desired behavioral outputs given specific users' inputs. To ensure the web front-end behaves as expected after code changes, we use Selenium combined with the `Pytest-Flask` extension to simulate a user experience of navigating `Marvin` web pages and interacting with any element on the web page. These tests can be run locally or hooked into a CI service. `BrowserStack` is a service to run web front-end tests within different versions of multiple browsers, on any operating system, including tablets and mobile devices. We have connected Travis and `BrowserStack` together, so the web front-end tests Travis performs are sent to be run on the `BrowserStack` servers.

8. SUMMARY AND FUTURE POTENTIAL

We have presented the first public release of the `Marvin` software, a suite of tools for interacting with the SDSS-IV MaNGA data set. We have described the core components of `Marvin` in Section 2, the main one being the Multi-Modal Data Access System (Section 2.3). The MMA system is a novel method of delivering MaNGA data to the user in an agnostic manner, seamlessly switching between local files and a remote database. This allows the user to focus more on their scientific analysis with the data and less on the overheads of data access and correct data integration into existing scripts. The MMA is tightly integrated into a suite of Python Tools (Section 3) designed to streamline the users' interaction with the core MaNGA data deliverables, and provide remote querying (Section 3.3) against the entire MaNGA data set from within a Python terminal, eliminating the need to download the full sample. `Marvin`'s `Query` system utilizes a pseudo-

natural language syntax parser (Section 3.3.1) to simplify the writing of SQL queries down to the most meaningful component, namely science-pertinent conditionals. `Marvin` provides a web front-end (Section 4), allowing for quick visual exploration of the MaNGA data set, presented in an intuitive, clean manner that lowers the barrier of entry for new users into MaNGA data. The `Marvin` team of developers is fully committed to an open-source model of development and has adopted modern best practices to ensure long-term software sustainability (Section 7). We invite members of the community to adopt the `Marvin` software in their workflows for scientific analysis of MaNGA data, to provide insightful feedback and report issues, or to contribute new features and functionality.

`Marvin` has potential in the future to grow to serve not only the existing community of MaNGA users, but also the overall astronomy community. In Section 6 we described a typical workflow with the MaNGA data set. However, this workflow is not really unique to MaNGA. Replace the MaNGA data with any other astronomical data set and the workflow essentially remains the same. In the same way that `Marvin` addresses the inherent issues with the MaNGA workflow, a generalization of `Marvin` has the potential to address the same issues for other astronomy workflows. The most natural extension of `Marvin` is to other surveys producing IFU-based data products, but the design of `Marvin` and the `Brain` inherently make them applicable to generic data sets within astronomy, as well as other scientific disciplines.

8.1. *The `Brain`*

Astronomy has many logistical challenges and overheads that are often overcome by reinventing existing tools. There is no suite of tools providing the out-of-the-box capabilities to address these issues. Many of the underlying Python packages (e.g., `requests`, `Flask`, `SQLAlchemy`) that are often used to solve these challenges are difficult to learn for newcomers and do not provide a usable solution by themselves. A product is needed that combines these packages in a way that addresses the needs of large scientific data sets with complex interrelationships. The `Brain` is the beginning of this product. By further abstracting out `Marvin`'s building blocks into the `Brain`, the `Brain` will become a complete framework for a data distribution system, providing seamless connections between web components, APIs, and programmatic Python tools. Furthermore, as the backbone of `Marvin`, this abstraction will provide a more robust, sustainable framework for `Marvin` that will make it even easier to extend to other data sets.

8.2. *As a Template*

`Marvin` serves as an example of a data distribution system but currently is quite specific to the MaNGA data set with nuggets of generalization scattered throughout it. With the proper abstraction of the `Brain`, it will provide a customizable, "Blank Slate", template product to handle more complex data sets and become a reusable toolkit for other projects within astronomy and other disciplines. This product, when given a file and a database presentation of that file, will provide base classes to provide a connected environment surrounding that file, with local and remote access, programmatic tools, a remote query system, and a web front-end to the data with a minimum display view. Creating a functional application beyond the "Blank Slate" would simply involve building a new Python package based off the `Brain`; subclassing its base classes; and adding functionality and details necessary for their particular application. This template could be used for rapid deployment and distribution of data sets for any research group ranging in size from small local teams to large collaborations or surveys.

8.3. *`SciServer` Integration*

While `Marvin` works either as a local analysis package, or for browser-based visual exploration, it still requires local package installation for local analysis, and focuses on single user software usage. Local package installation can often interfere with custom user environments, while single user usage and analysis limits the ability for collaborative science. However, an advantage of `Marvin` is that it can be deployed either as a client service, or in a server mode distributing content, into existing archive systems or in different environments. To enable collaborative and remote scientific analysis, we are in the process of integrating `Marvin` into the `SciServer` platform. `SciServer` [11] is a fully integrated cyber-infrastructure system encompassing related, integrated, tools and services to enable researchers to cope with, and collaborate around, scientific big data.

`SciServer` integration will enable users to utilize the access and analysis capabilities of `Marvin` without having a local installation. Collaborative science will be enabled through remote, persistent, `Jupyter`-notebooks that can be shared amongst multiple users. Analysis can be offloaded to the `SciServer` system removing the need for any data to be hosted locally. Additionally, through the `SciServer` Compute system, the `Marvin Query` tool will be expanded to include asynchronous query capability, allowing intensive queries to be submitted as jobs, similar to the existing SDSS CasJobs system, freeing up the user's local terminal.

[11] http://www.sciserver.org/

8.4. *Community Driven Development*

Our goal with `Marvin` was to make it as easy as possible for users to interact with MaNGA data. In the pursuit of that goal, we have developed a software framework that is quite extensible, making it extremely easy to contribute tools, features, and functions that enhance `Marvin`'s capabilities. `Marvin` takes care of all logistical overhead of interacting with the data, as well as defining all core tools that provide most of the desired information. We realize, however, the greatest usefulness of `Marvin` are the things not yet done. While the developers have provided the initial scope of `Marvin` the community can greatly expand its usefulness through code contributions. Through the open source nature of `Marvin` and the ability to wrap the existing suite of tools, the community can easily provide new functionality to aid in the overall scientific usefulness of the MaNGA data set for people at any stage of their scientific career.

8.5. *User Metrics and Broader Impact*

To properly understand the benefit of software within the community, and to better aid in future software funding, it is of paramount importance that software developers play a more active role in assessing the impact and usefulness of their software within the community. As software itself plays an often overlooked but important role in the pursuit of scientific discovery, developers not only have a role to provide to the community but also to better understand their users for future developments that aid in that discovery. We aim to address exactly these issues. Marvin has been operating in Beta for over a year now. In this time we have been collecting anonymous user statistics and metrics (e.g. web/API route access or query submissions), and obtaining feedback on use cases for users within the collaboration. With this data, plus the data we will collect with the public release of `Marvin`, we have a good opportunity to assess the impact of `Marvin` on the community of users. We hope to provide feedback on the current usage statistics for `Marvin`, some general examples of scientific workflows utilizing `Marvin`, as well as lessons learned throughout the development of `Marvin`.

We would like to acknowledge the MaNGA team for supporting this project, helping shape its design, and providing critical feedback and testing during various shakedown phases. We would like to thank Demitri Muna for his help with `Flask` and database services. We give thanks to 8485-1901 for its noble sacrifice as our guinea pig.

Funding for the Sloan Digital Sky Survey IV has been provided by the Alfred P. Sloan Foundation, the U.S. Department of Energy Office of Science, and the Participating Institutions. SDSS-IV acknowledges support and resources from the Center for High-Performance Computing at the University of Utah. The SDSS web site is www.sdss.org.

SDSS-IV is managed by the Astrophysical Research Consortium for the Participating Institutions of the SDSS Collaboration including the Brazilian Participation Group, the Carnegie Institution for Science, Carnegie Mellon University, the Chilean Participation Group, the French Participation Group, Harvard-Smithsonian Center for Astrophysics, Instituto de Astrofísica de Canarias, The Johns Hopkins University, Kavli Institute for the Physics and Mathematics of the Universe (IPMU) / University of Tokyo, the Korean Participation Group, Lawrence Berkeley National Laboratory, Leibniz Institut für Astrophysik Potsdam (AIP), Max-Planck-Institut für Astronomie (MPIA Heidelberg), Max-Planck-Institut für Astrophysik (MPA Garching), Max-Planck-Institut für Extraterrestrische Physik (MPE), National Astronomical Observatories of China, New Mexico State University, New York University, University of Notre Dame, Observatário Nacional / MCTI, The Ohio State University, Pennsylvania State University, Shanghai Astronomical Observatory, United Kingdom Participation Group, Universidad Nacional Autónoma de México, University of Arizona, University of Colorado Boulder, University of Oxford, University of Portsmouth, University of Utah, University of Virginia, University of Washington, University of Wisconsin, Vanderbilt University, and Yale University.

Software: Anaconda (https://anaconda.org/anaconda/python), Astropy (Astropy Collaboration et al. 2013; The Astropy Collaboration et al. 2018, http://www.astropy.org), Bootstrap (https://getbootstrap.com), Browserstack (https://www.browserstack.com) brain (https://github.com/sdss/marvin_brain), Coveralls (https://coveralls.io/), D3 (https://d3js.org), DyGraphs (http://dygraphs.com), FITS (Pence et al. 2010), Flask (http://flask.pocoo.org), Flask-Login (https://flask-login.readthedocs.io), Flask-JWT-Extended (https://flask-jwt-extended.readthedocs.io), fuzzywuzzy (https://github.com/seatgeek/fuzzywuzzy), git (https://git-scm.com), Highcharts (https://www.highcharts.com), Jinja2 (http://jinja.pocoo.org/docs), JQuery (https://jquery.com), Jupyter (Kluyver et al. 2016, http://jupyter.org), Matplotlib (Hunter 2007, https://doi.org/10.5281/zenodo.61948), networkx (https://networkx.github.io), Nginx (https://www.nginx.com), OpenLayers (https://openlayers.org), pip (https://pypi.org/project/pip), Postgres (https://www.postgresql.org),

pytest (https://docs.pytest.org/), Read the Docs (https://readthedocs.org/), requests (http://docs.python-requests.org), rsync (https://rsync.samba.org), sdss-access (https://doi.org/10.5281/zenodo.1410704), sdss-tree (https://doi.org/10.5281/zenodo.1410706), Selenium (https://www.seleniumhq.org), Sphinx (http://www.sphinx-doc.org), SQLAlchemy (https://www.sqlalchemy.org), sqlalchemy-boolean-search (https://github.com/sdss/sqlalchemy-boolean-search), Travis-CI (https://travis-ci.org/), uwsgi (https://uwsgi-docs.readthedocs.io)

REFERENCES

Astropy Collaboration, Robitaille, T. P., Tollerud, E. J., et al. 2013, A&A, 558, A33

Baldwin, J. A., Phillips, M. M., & Terlevich, R. 1981, PASP, 93, 5

Belfiore, F., Maiolino, R., Maraston, C., et al. 2016, MNRAS, 461, 3111

Bershady, M. A., Verheijen, M. A. W., Swaters, R. A., et al. 2010, ApJ, 716, 198

Blanton, M. R., Bershady, M. A., Abolfathi, B., et al. 2017, AJ, 154, 28

Blanton, M. R., Kazin, E., Muna, D., Weaver, B. A., & Price-Whelan, A. 2011, AJ, 142, 31

Braun, R., Bourke, T., Green, J. A., Keane, E., & Wagg, J. 2015, Advancing Astrophysics with the Square Kilometre Array (AASKA14), 174

Bundy, K., Bershady, M. A., Law, D. R., et al. 2015, ApJ, 798, 7

Cappellari, M., Emsellem, E., Krajnović, D., et al. 2011, MNRAS, 413, 813

Cherinka, B. & Brownstein, J. R. 2018, sdss/tree: tree, Zenodo

Cherinka, B., Brownstein, J. R., & Blanton, M. 2018, sdss/sdss-access: sdss-access, Zenodo

Drory, N., MacDonald, N., Bershady, M. A., et al. 2015, AJ, 149, 77

Gunn, J. E., Siegmund, W. A., Mannery, E. J., et al. 2006, AJ, 131, 2332

Hunter, J. D. 2007, Computing In Science & Engineering, 9, 90

Ivezić, Ž., Kahn, S. M., Tyson, J. A., et al. 2008, ArXiv e-prints

Kluyver, T., Ragan-Kelley, B., Pérez, F., et al. 2016, in ELPUB, 87–90

Law, D. R., Cherinka, B., Yan, R., et al. 2016, AJ, 152, 83

Law, D. R., Yan, R., Bershady, M. A., et al. 2015, AJ, 150, 19

Moore, G. E. 1965, Electronics, 38

Nielsen, J. 1998, Nielsen's Law of Internet Bandwidth, https://www.nngroup.com/articles/law-of-bandwidth, accessed: 2018-08-13

Pence, W. D., Chiappetti, L., Page, C. G., Shaw, R. A., & Stobie, E. 2010, A&A, 524, A42

Sánchez, S. F., Kennicutt, R. C., Gil de Paz, A., et al. 2012, A&A, 538, A8

Smee, S. A., Gunn, J. E., Uomoto, A., et al. 2013, AJ, 146, 32

Strauss, M. A., Weinberg, D. H., Lupton, R. H., et al. 2002, AJ, 124, 1810

The Astropy Collaboration, Price-Whelan, A. M., Sipőcz, B. M., et al. 2018, ArXiv e-prints

Wake, D. A., Bundy, K., Diamond-Stanic, A. M., et al. 2017, AJ, 154, 86

Weaver, B. A., Blanton, M. R., Brinkmann, J., Brownstein, J. R., & Stauffer, F. 2015, Publications of the Astronomical Society of the Pacific, 127, 397

Yan, R., Bundy, K., Law, D. R., et al. 2016, The Astronomical Journal, 152, 197

Yan, R., Tremonti, C., Bershady, M. A., et al. 2016, AJ, 151, 8

York, D. G., Adelman, J., Anderson, Jr., J. E., et al. 2000, AJ, 120, 1579

APPENDIX

A. EXAMPLE CODE

A.1. *Illustration of the* `Marvin` *MMA*

This illustrates the access of a `Marvin Cube` object, utilizing the `Marvin` MMA, in the Python programming language. After the initial import of the `Marvin Cube` tool, a DRP cube is accessible by explicitly providing as input either the full path to a local file or by an object ID. When explicitly specifying a filename (first example), `Marvin` assumes a **mode=local** and a **data_origin=file**. When specifying an object ID, `Marvin` will first attempt to open the object locally from a local database or from a file if one is found (second example). If an object cannot be found locally (third example), `Marvin` switches to **mode=remote** and **data_origin=api** to retrieve the object from a remote location. In each of the cases, the output is the same, an instance of the `Marvin` Cube which wraps the specified DRP datacube.

```
from marvin.tools.cube import Cube
# INFO: No release version set. Setting default to MPL-6

# access locally by filename
c = Cube("/Users/Brian/manga/redux/v2_3_1/8485/stack/manga-8485-1901-LOGCUBE.fits.gz")
# <Marvin Cube (plateifu="8485-1901", mode="local", data_origin="file")>

# access locally by id, will either use a database
c = Cube("8485-1901")
# <Marvin Cube (plateifu="8485-1901", mode="local", data_origin="db")>
# ...or a file
# <Marvin Cube (plateifu="8485-1901", mode="local", data_origin="file")>
```

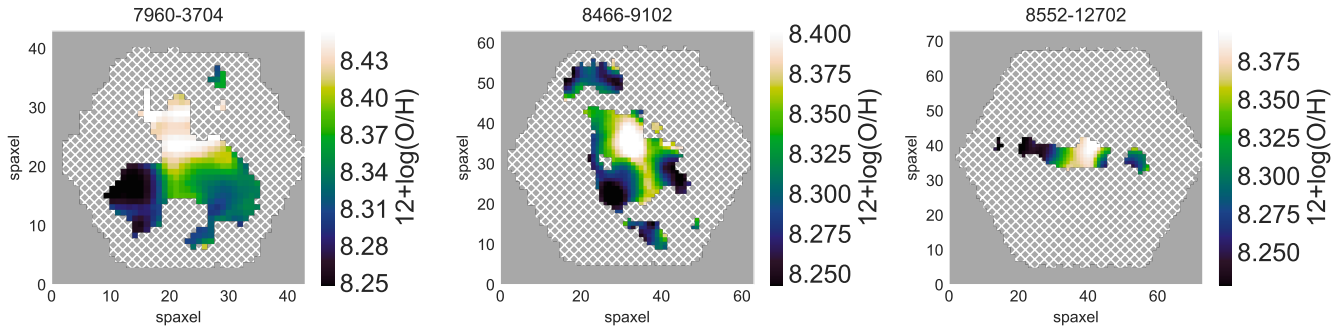


Figure A1. Gas-phase metallicity maps for three galaxies (7960-3704, 8466-9102, and 8552-12702) with only star-forming spaxels shown.

```
# access remotely when no local version found
c = Cube("8485-1902")
# WARNING: local mode failed. Trying remote now.
# <Marvin Cube (plateifu="8485-1902", mode="remote", data_origin="api")>
```

A.2. *Map Plotting*

The following code was used to produce the right panel of Figure 1, which a 2×2 multi-panel plot of Hα flux, log([N II] λ6585 / Hα) flux, stellar velocity, and stellar velocity dispersion corrected for instrumental broadening. This snippet illustrates remote data access for retrieving the maps, map arithmetic and applying logarithms, and the simple but customizable plotting method for `Maps`.

```
import matplotlib.pyplot as plt
import numpy as np

from marvin.tools import Maps

maps = Maps("7977-12705")

halpha = maps.emline_gflux_ha_6564
nii_ha = np.log10(maps.emline_gflux_nii_6585 / halpha)
stvel = maps.stellar_vel
stsig = maps.stellar_sigma
stsig_corr = stsig.inst_sigma_correction()

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 11))
halpha.plot(fig=fig, ax=axes[0, 0])
nii_ha.plot(fig=fig, ax=axes[0, 1], title="log([NII]6585 / H-alpha)", snr_min=None)
stvel.plot(fig=fig, ax=axes[1, 0])
stsig_corr.plot(fig=fig, ax=axes[1, 1])
```

A.3. *Metallicity Science Use Case*

The following code computes and plots gas-phase metallicity maps (Figure A1) and gradients (Figure A2) for star-forming spaxels only. The code selects galaxies based on global parameters using a `Marvin` query. It then generates masks to select the star-forming spaxels using `Marvin`'s BPT tool (see Section 3.4). Finally it takes advantage of `Marvin`'s map arithmetic to compute metallicities.

```
from matplotlib import cm
import matplotlib.pyplot as plt
import numpy as np

from marvin.tools.query import Query
import marvin.utils.plot.map as mapplot

# Define query and run it
searchfilter = ("nsa.sersic_logmass >= 9 and "
                "nsa.sersic_logmass <= 9.1 and "
                "nsa.elpetro_mag_g_r < 0.3")
```

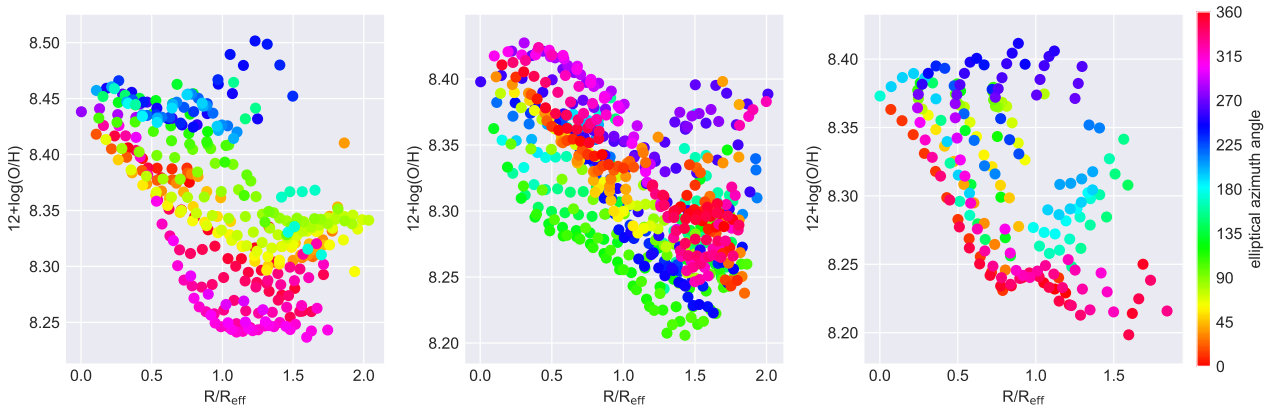


Figure A2. Radial gas-phase metallicity gradients (in units of effective radius $R_{\rm eff}$) for three galaxies (7960-3704, 8466-9102, and 8552-12702) color-coded by elliptical azimuthal angle with only star-forming spaxels shown.

```
query = Query(searchfilter=searchfilter, limit=3)
results = query.run()

# Convert query results into a list of Marvin data product objects
results.convertToTool("maps")
galaxies = results.objects

metallicity = []
for it in galaxies:

    # Lazy-load [NII] and Halpha maps for each galaxy
    nii = it.emline_gflux_nii_6585
    halpha = it.emline_gflux_ha_6564

    # Compute metallicity, 12+log(O/H), using Eq. 1 from Pettini & Pagel (2004)
    N2 = np.log10(nii / halpha)
    oh = 8.90 + 0.57 * N2
    metallicity.append(oh)

# Create BPT masks for each galaxy
bpt_masks = [it.get_bpt(return_figure=False, show_plot=False) for it in galaxies]

# Default bad data labels for MANGA_DAPPIXMASK
labels_bad = ["NOCOV", "UNRELIABLE", "DONOTUSE"]

# Apply BPT cut to select star-forming spaxels and mask out spaxels with bad data
masks = []
for metal, bpt in zip(metallicity, bpt_masks):
    # Mask out non-star-forming spaxels
    starforming = bpt["sf"]["global"]
    non_sf = ~starforming * metal.pixmask.labels_to_value("DONOTUSE")

    # Merge the non-star-forming mask with the MANGA_DAPPIXMASK mask
    masks.append(metal.mask | non_sf)

# Metallicity maps
fig, axes = plt.subplots(1, 3, figsize=(12, 3))
for ax, metal, mask, gal in zip(axes, metallicity, masks, galaxies):
    mapplot.plot(dapmap=metal, mask=mask, fig=fig, ax=ax,
                 cblabel="12+log(O/H)", title=gal.plateifu)

fig.tight_layout()

# Metallicity gradients
fig, axes = plt.subplots(1, 3, figsize=(12, 4))
plt.subplots_adjust(left=0.08, wspace=0.25)
for ax, metal, mask, gal in zip(axes, metallicity, masks, galaxies):
```

```
    mappable = ax.scatter(
        gal.spx_ellcoo_r_re.value[~mask.astype(bool)],
        metal.value[~mask.astype(bool)],
        c=gal.spx_ellcoo_elliptical_azimuth.value[~mask.astype(bool)],
        cmap=cm.hsv,
        vmin=0,
        vmax=360,
    )
    ax.set_xlabel("R/R$_\mathrm{eff}$")
    ax.set_ylabel("12+log(O/H)")
    ax.set_title(gal.plateifu)

cax = fig.add_axes([0.91, 0.105, 0.01, 0.78])
ticks = np.linspace(0, 360, 9)
cb = fig.colorbar(mappable, cax, ticks=ticks)
cb.set_label("elliptical azimuth angle")
```

B. Marvin TUTORIALS

In addition to installation instructions and detailed descriptions of the Marvin components, the online documentation also contains tutorials[12], including exercises on specific science cases and example Jupyter notebooks. The tutorials are intended to familiarize the user with working with MaNGA data via Marvin. Each tutorial is designed as a series of steps that build on top of one another and flow together to provide a layered learning experience, guiding the user from a straightforward approach to more in-depth Marvin usage. These tutorials act as a good entry point for new users diving into MaNGA data.

A few example Marvin tutorials are:

- Lean Tutorial[13]: an example project using Marvin from start to finish. It focuses on the calculation of the [N II]/Hα ratio for star-forming spaxels in galaxies with stellar mass $\log(M_\star) = 10$–11 $M_\odot$.
- Plotting Tutorial[14]: plotting examples of increasing complexity from a single map or spectrum to heavily customized multi-panel map plots.
- Sample Selection Tutorial[15]: examples of how to search on the Primary, Secondary, and Color-Enhanced samples that make up the MaNGA main sample (Yan et al. 2016; Wake et al. 2017).

Additional Marvin tutorials are available on the main SDSS DR15 site[16]. These tutorials are designed as simple introductions to a few basic features of Marvin, with more advanced tutorials in the main Marvin documentation. Tutorials are provided for both Marvin's front-end web interface, as well as the Tools. The Tools tutorials focus on accessing spectra from individual spaxels, plotting maps for derived analysis properties with custom masks, identifying unique bins, and extracting a binned spectrum. For the web tutorials, there are two videos provided to highlight some relevant features. One video focuses on exploring an individual MaNGA galaxy. It highlights basic information, the dynamic interactive maps and spectral visualization, and detailed information from the NSA catalog. The second video focuses on performing queries on the MaNGA dataset. It explains how to use the simplified SQL interface, the guided SQL builder, and the table of search results. Both video-tutorials are subtitled in English.

[12] https://sdss-marvin.readthedocs.io/en/stable/tutorials.html

[13] https://sdss-marvin.readthedocs.io/en/stable/tutorials/lean-tutorial.html

[14] https://sdss-marvin.readthedocs.io/en/stable/tutorials/plotting-tutorial.html

[15] https://sdss-marvin.readthedocs.io/en/stable/tutorials/sample-selection.html

[16] https://www.sdss.org/dr15/manga/manga-tutorials/marvin-tutorial/