mycoNet2

A system for automatically training gaussian process machine learning models on mycology data using Python and MySQL

**SYSTEM REFERENCE DOCUMENT**

# Contents

[Contents 2](#_Toc109800582)

[System Overview 4](#_Toc109800583)

[Scheduled Tasks 4](#_Toc109800584)

[Python 3.10.5 4](#_Toc109800585)

[PHP 8 5](#_Toc109800586)

[Apache Web Server 5](#_Toc109800587)

[phpMyAdmin 5](#_Toc109800588)

[MySQL Server 8.0 5](#_Toc109800589)

[mycoNet2 Code 6](#_Toc109800590)

[Source Directory 6](#_Toc109800591)

[mycologyHelpers.py 6](#_Toc109800592)

[DBConnection.py 6](#_Toc109800593)

[GPFactory.py 7](#_Toc109800594)

[train.py 7](#_Toc109800595)

[trainModels.bat 7](#_Toc109800596)

[Output Directory 7](#_Toc109800597)

[\*.dump 7](#_Toc109800598)

[logs directory 7](#_Toc109800599)

[Config Directory 7](#_Toc109800600)

[index.json 7](#_Toc109800601)

[config.json 8](#_Toc109800602)

[Appendix A: mycologyHelpers.py 9](#_Toc109800603)

[Variables 9](#_Toc109800604)

[outputDir 9](#_Toc109800605)

[configDir 9](#_Toc109800606)

[logState 10](#_Toc109800607)

[Functions & Classes 10](#_Toc109800608)

[Logger 10](#_Toc109800609)

[toggleLog() 10](#_Toc109800610)

[crc32Opt(filepath: str) 10](#_Toc109800611)

[getDateTime() 10](#_Toc109800612)

[readJSON(whichDir: str, filename: str) 10](#_Toc109800613)

[initConfig() 10](#_Toc109800614)

[dumpConfig(config: dict) 10](#_Toc109800615)

[dictChecksum(dict: dict) 10](#_Toc109800616)

[envChecksum() 10](#_Toc109800617)

[Appendix B: DBConnection.py 10](#_Toc109800618)

[DBConnection 10](#_Toc109800619)

[\_\_getSQLCursor(self: Self, username: str, password: str) 11](#_Toc109800620)

[SQLQuery(self: Self, query: str) 11](#_Toc109800621)

[Appendix C: GPFactory.py 11](#_Toc109800622)

[GPFactory 11](#_Toc109800623)

[\_\_extractConfig(self: Self, whichMold: int) 11](#_Toc109800624)

[\_\_generateOutput(self: Self, model: GaussianProcessRegressor, whichMold: int) 11](#_Toc109800625)

[\_\_getSpec(self:Self, whichMold: int) 11](#_Toc109800626)

[\_\_pushModelResults(self: Self, model: GaussianProcessRegressor, whichMold: int) 11](#_Toc109800627)

[\_\_updateMetaData(self: Self, whichMold: int) 11](#_Toc109800628)

[\_\_dumpTrainedModel(self: Self, model: GaussianProcessRegressor, whichMold: int) 11](#_Toc109800629)

[\_\_getDataChecksum(self: Self, whichMold: int) 12](#_Toc109800630)

[dataChangedSinceLastTrain(self: Self, whichMold: int) 12](#_Toc109800631)

[trainModel(self: Self, whichMold: int) 12](#_Toc109800632)

[Appendix D: train.py 12](#_Toc109800633)

[Functions & Classes 12](#_Toc109800634)

[trainOne(whichMold: int) 12](#_Toc109800635)

[train() 12](#_Toc109800636)

# **System Overview**

The mycoNet2 system is hosted on the MYC1.greerlabs.com virtual machine and is comprised of the following interconnected pieces of software/tools:

* MySQL Server 8.0
  + Hosts the mycology database used to store mold data and machine learning predictions.
* phpMyAdmin
  + Provides a graphical user interface to the mycology database, allowing the mycology team to administrate and use their own database without knowledge of SQL.
  + Accessible over greerlabs.com network at myc1.greerlabs.com/phpMyAdmin.php
  + Location: C\webserver\Apache24\htdocs\phpmyadmin
* Apache Web Server
  + Hosts the phpMyAdmin page on the local greerlabs.com network.
  + Location: C:\webserver\Apache24
* Python 3.10.5
  + Used to run the mycoNet2 code.
* PHP 8
  + Required for the proper operation of phpMyAdmin
  + Location: C:\webserver\PHP
* mycoNet2 Code
  + Provides helper functions for connecting to the database and pulling/pushing relevant data.
  + Provides code for training and querying trained machine learning models.
  + Location: C:\query\mycoNet2

The server requires a network connection to the greerlabs.com company network, but does not require any external network connection, therefore can be safely disconnected from the internet if there are any security concerns. The server will deny any incoming connections from outside of the local network, and only responds to the hostname myc1.greerlabs.com, denying requests formatted as an IP address or localhost.

# Scheduled Tasks

The mycoNet2 system performs nightly retraining of the machine learning models based on updated data, scheduled through Task Scheduler. This task runs the trainModels.bat file from the source directory every night at 11:59:59 PM, with permissions to run without any users logged in, and waking the machine to do so if necessary. This is not strictly necessary to the function of the system, but it is recommended to keep all machine learning predictions up to date with the latest data. If it is decided this task is to be removed, it will be the responsibility of the users to manually run the trainModels.bat file to update the machine learning predictions.

# Python 3.10.5

Python 3.10.5 has been installed on the server in the default install location, as well as several packages, all of which are necessary for proper function of the mycoNet2 code. The mycoNet2 code contains a Python virtual environment in the config/venv directory, allowing for quick reinstallation of all dependencies should a system migration be necessary.

# PHP 8

PHP 8 has been installed on the server to provide the required functionality for phpMyAdmin to function. This is a portable installation and can be easily moved to a new system in the event of a migration.

# Apache Web Server

Apache Web server has been installed in the C:\webserver directory. This is a portable installation and can be easily moved to a new system in the event of a migration.

# phpMyAdmin

phpMyAdmin is a graphical, PHP-based interface to the SQL database, hosted on the local network using Apache web server. Users can connect to the SQL database through phpMyAdmin by going to myc1.greerlabs.com while connected to the company network. This is a standalone interface and is not necessary for the proper operation of the mycoNet2 code, rather, its purpose is to give the mycology department easy access to view their database in a graphical fashion.

# MySQL Server 8.0

MySQL server is installed in the default install location and is used to host the databases for the mycology department. The mycoNet2 code expects the existence of a database called “mycology,” which should have at minimum, the following tables and columns:

* mold\_lots
  + mold\_id
  + incubation\_days
  + seed\_days
  + plate\_days
  + yield\_per\_liter
* specs
  + incubation\_days\_min
  + incubation\_days\_max
  + seed\_days\_min
  + seed\_days\_max
  + plate\_days\_min
  + plate\_days\_max
* gp\_predictions
  + mold\_id
  + incubation\_days
  + seed\_days
  + plate\_days
  + predicted\_average\_yield\_per\_liter
  + std\_deviation

The current implementation of the database includes several other tables and columns for ease of information access and additional optional filtering of data, but these additional tables are not required and the mycoNet2 code will run without them if necessary.

NOTE: If the mold\_lots table is reduced to the bare minimum listed above, the conditionals subkey of the default key in the [config.json](#_config.json) file will have to be set to an empty list, as it references some of these optional columns.

# mycoNet2 Code

The current version of the code enforces and expects the following directory structure:

* mycoNet2/
  + source/
    - mycologyHelpers.py
    - GPFactory.py
    - DBConnection.py
    - train.py
    - trainModels.bat
  + output/
    - logs/
      * training\_log\_\*.txt
    - \*.dump
  + config/
    - venv/
      * requirements.txt
      * pyvenv.conf
      * <python packages>
    - config.json
    - index.json

All file paths used in the code are referenced relative to the location of the source directory, and any missing files and folders except for config/venv will be regenerated if the source folder is intact

## Source Directory

The source directory contains all python source code for the mycoNet2 system, a batch file for use with Task Scheduler, and an optional compiled Windows binary if installing Python on a new system if a data migration is not feasible.

### mycologyHelpers.py

This file contains helper functions and classes for connecting to and querying the database, logging, reading and writing necessary files, and formatting date/time strings. It also enforces and expects the directory structure listed above. This code exists purely to be imported into other scripts, so it should not be modified to execute any code. A description of functions and variables included in this file can be found in [Appendix A.](#_Appendix_A:_mycologyHelpers.py)

### DBConnection.py

This file contains a class definition for an object that stores a connection to the SQL database and processes queries through the stored connection. This code exists purely to be imported into other scripts, so it should not be modified to execute any code. A description of functions and variables included in this file can be found in [Appendix B.](#_Appendix_B:_DBConnection.py)

### GPFactory.py

This file contains a class definition using the Singleton Factory software engineering design pattern. This Factory class encapsulates all necessary behavior for communicating with the MySQL database and training the GaussianProcessRegressor models. This code exists purely to be imported into other scripts, so it should not be modified to execute any code. A description of functions and variables included in this file can be found in [Appendix C.](#_Appendix_C:_GPFactory.py)

### train.py

This file contains functions that instantiate instances of GPFactory for training of GaussianProcessRegressor models. It is also the file which is executed by trainModels.bat. The executable portion of this file contains a single function call to [train()](#_train()). A description of functions and variables included in this file can be found in [Appendix D.](#_Appendix_D:_train.py)

### trainModels.bat

This batch file runs train.py and is used to allow any necessary modifications to the scheduled run process to be made without needing to alter the task in Task Scheduler.

## Output Directory

### \*.dump

Files with the .dump extension are locally saved copies of trained GaussianProcessRegressor objects. These files are automatically named my{mold id}\_gauss.dump, but can be renamed if desired, so long as the corresponding entry in [index.json](#_index.json) is changed as well, otherwise the program will perceive the file as missing and retrain the model. These dumps are created through the pickle library and can be loaded into other programs using the same library, although the exact details of how to do so are out of the scope of this documentation and will not be included here.

### logs directory

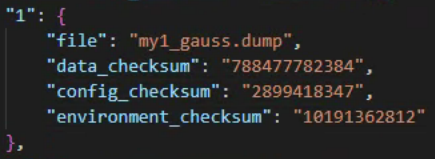
This directory contains all the log files generated from the training of the gaussian process models. Each log file is named based on the date that training was started and contains all of the training logs from that day. If training is run multiple times in one day, all the logging data across all that day’s training runs will be stored in one file. The log files contain a few lines of text for each mold ID. If a line states that training is successful, it means that there have been no issues, and the two subsequent lines will also pertain to that mold, with one giving extra information about the resulting kernel parameters from the training session, and the other giving information about the success or failure of uploading trained results to the database. If a line states that training is aborted, it will list a reason, either due to insufficient data or a lack of updated data since the last train. If for some reason the task is interrupted in the middle of training or uploading, it will not be marked as completed in the index file and the train/upload process will be restarted the next time training is initiated.

## Config Directory

The config directory stores all the configuration and persistent data necessary for the proper execution of the program.

### index.json

This file is automatically generated by the system as new models are trained and contains data pertaining to the trained models. Each main entry in the file is the string representation of a mold ID. MY1 has an ID of 1, MY2 has an ID of 2, etc. These IDs are defined in the molds table of the mycology MySQL database. Each entry then has four values: file, data\_checksum, config\_checksum, and env\_checksum. The file value stores the name of the .dump file associated with that particular mold. The other keys store data used to determine whether anything has changed since the last time the gaussian process model for that mold was changed. This prevents needless retraining. In general, it is best to not modify this file, as causing it to become unreadable or malformed will result in the whole suite of models being retrained. If for some reason a .dump file needs to be renamed, be sure to update its corresponding entry in index.json, otherwise the system will perceive the file as missing.



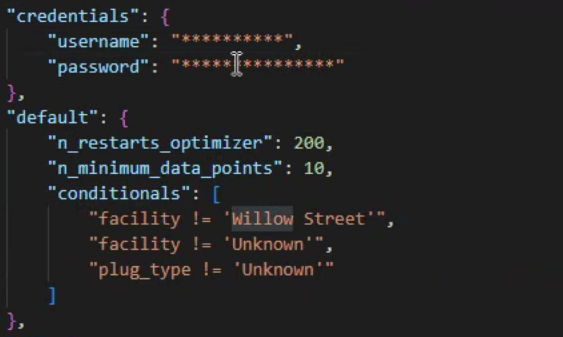
Sample entry in index.json

### config.json

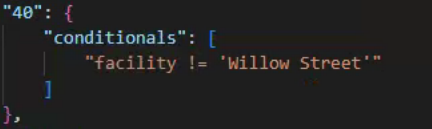
This file contains parameters governing how each gaussian process model is trained. It is not recommended to modify anything in this file without a solid understanding of how the mycoNet2 system works. This file requires at least two main entries, credentials and default.

Credentials stores the username and password used for connecting to the database. By default, it uses the “python” user which is created to have only the exact permissions necessary for proper operation of the program. Use caution when changing the credentials to those of a user with elevated permissions.

The default entry stores the default training behavior for all gaussian process models. This entry requires 3 subkeys, n\_restarts\_optimizer, n\_minimum\_data\_points, and conditionals. n\_restarts\_optimizer determines how many times the optimizer is restarted to find the best possible fit given the data. This setting has a direct impact on the training time for each mold. Higher values mean better accuracy at the cost of additional training time, while the opposite is true for lower values. n\_minimum\_data\_points determines how many data points are required at minimum for a model to be allowed to train on a given mold. Conditionals should be a list of MySQL formatted conditional modifiers. These modifiers are appended to the end of a SELECT statement used to retrieve a mold’s data from the database. These modifiers can be used to exclude unwanted data from training. By default, all data from non-Lenoir facilities and those with an unknown plug type are excluded.



In addition to this, additional entries may be added below the default entry to override specific training behaviors for specific molds. In this event, the key should be the mold’s ID, and only subkeys which differ from the default entry need to be specified.



WARNING: mycoNet2 does NOT sanitize inputs from the config file, so ensure that formatting is correct before running the program. It is recommended that only users trained in SQL are allowed to modify this file, and any requests to modify training behavior be given to a trained user. It is also discouraged to change what is stored in the credentials entry, as the provided user account protects against harmful queries by having read-only permissions to most of the database. In addition, changes to the default entry or the creation of mold-specific entries will directly impact the accuracy of trained gaussian process models. In general, this file should be fine as it is and should not be modified unless absolutely necessary. This file and the source code are protected from general modification by being accessible only through a direct connection to the host server.

## Appendix A: mycologyHelpers.py

#### Variables

##### outputDir

Defines location of output directory relative to source directory.

##### configDir

Defines location of config directory relative to source directory.

##### logState

Global variable that records whether logging is currently enabled or disabled.

#### Functions & Classes

##### Logger

Defines an object which duplicates stdout between the terminal and a predefined log file, whose location is determined by the outputDir variable. The name of the log file is determined based on the current date at the time the Logger object was instantiated.

##### toggleLog()

When logState is false, creates a new Logger object and sets logState to true. When logState is true, stdout is reset to normal and any existing Logger objects are destroyed.

##### crc32Opt(filepath: str)

Uses the zlib.crc32 function to generate CRC32 checksum of the file located at filepath. Performance optimized to read file in chunks.

##### getDateTime()

Returns a string containing “{current date} at {current time}”. Used to generate headings for log files.

##### readJSON(whichDir: str, filename: str)

Reads in the JSON file named filename in directory whichDir into a dictionary. If there are any errors when reading the file, this function silently returns an empty dictionary. If the file being read is specifically the config.json file, any error in reading will terminate the program and display an error message, since this file is required for the program to run correctly.

##### initConfig()

If the configuration file or any of its required parts are missing, a warning message will be displayed, and the configuration will be reset to the defaults. This function will overwrite any changes to the config.json file, so all changes will be lost. Ensure that the file contains at least a credentials field with username and password values, and a default field with n\_restarts\_optimizer, n\_minimum\_data\_points, and conditionals. If these are present, initConfig will do nothing.

##### dumpConfig(config: dict)

Dumps the dictionary provided as an argument to the config.json file.

##### dictChecksum(dict: dict)

Returns the CRC32 checksum of the provided dictionary.

##### envChecksum()

Returns the CRC32 checksum of the program’s environment. This includes the python virtual environment, and the source code files GPFactory.py, DBConnection.py, and mycologyHelpers.py.

## Appendix B: DBConnection.py

#### DBConnection

This class stores a connection and cursor object, which are used to query the SQL database, as well as to safely close the connection when the program terminates.

##### \_\_getSQLCursor(self: Self, username: str, password: str)

Attempts to use the mysql.connector library to establish a connection to the database mycology hosted on a local MySQL server, signing in as user username with password password. On an error, the function will print an error message describing the issue and terminate the program. On a success, the cursor object is defined to return dictionaries as responses from the server, and the connection and cursor objects are stored into the class variables.

##### SQLQuery(self: Self, query: str)

Uses the stored cursor object to run the specified query against the database, and returns the dictionary provided by the cursor.

## Appendix C: GPFactory.py

#### GPFactory

The GPFactory class defines a singleton factory object that contains all the algorithms for training models and interfacing with the local file system.

##### \_\_extractConfig(self: Self, whichMold: int)

Combines the default configuration behavior with the configuration modifiers specified for whichMold in the config.json file, then returns the new configuration settings.

##### \_\_generateOutput(self: Self, model: GaussianProcessRegressor, whichMold: int)

Defines the algorithm for outputting the results of a single model training for whichMold. Calls \_\_dumpTrainedModel(), \_\_updateMetaData() and \_\_pushModelResults in sequence.

##### \_\_getSpec(self:Self, whichMold: int)

Connects to the SQL database and gets the spec limits for incubation, seed, and plate days for mold number whichMold. Returns the spec as a dictionary.

##### \_\_pushModelResults(self: Self, model: GaussianProcessRegressor, whichMold: int)

Begins by clearing out all rows from the gp\_predictions SQL table corresponding to mold whichMold. Generates all possible combinations of incubation, seed, and plate days within the spec for mold number whichMold, then queries the GaussianProcessRegressor model on each combination. Runs an SQL query to insert a new row to the gp\_predications table for each query.

##### \_\_updateMetaData(self: Self, whichMold: int)

Updates the entry in the index.json file for mold number whichMold. Calls \_\_getDataChecksum to get the latest checksum of mold lot data for the given mold, and dictChecksum on the configuration settings for the given mold. Updates the dictionary entry for whichMold with the new data, then outputs the updated dictionary to the file.

##### \_\_dumpTrainedModel(self: Self, model: GaussianProcessRegressor, whichMold: int)

Uses the pickle package to convert a trained GaussianProcessRegressor to a storeable format, then dumps the raw data to a file named my{whichMold}\_gauss.dump in the output directory to store for later use. These pickled models can be loaded back into other programs if necessary, and are stored for archival purposes, as they are not strictly necessary to the function of the program. However, they serve as a backup if the SQL server ever goes offline or if the program needs to be converted away from SQL.

##### \_\_getDataChecksum(self: Self, whichMold: int)

Connects to the SQL database and returns a CRC32 checksum for the lot data corresponding to mold number whichMold.

##### calculateLotWeeks(self: Self, whichMold: int, incubationDays: int, predictedYield: int)

Calculates the lot\*weeks performance metric for a given mold based on the number of incubation days and predicted yield. The lot\*weeks formula is:

* X = yield/vessel
* Y = growth days
* K = Annual Demand / (# Vessels \* 7) = Arbitrary constant
* Lot\*Weeks = Y\*K/X

Contact Kevin Enck ([Kevin.Enck@stallergenesgreer.com](mailto:Kevin.Enck@stallergenesgreer.com)) for questions regarding this formula and its uses.

##### dataChangedSinceLastTrain(self: Self, whichMold: int)

Calculates the checksums for the mold lot data corresponding to whichMold as well as the configuration file and compares these to what is stored in the index file entry for whichMold. Also checks the environment checksum. If any of these checksums do not match what is stored in index.json, the data for the mold is considered different and the model is retrained on the most up to date data.

##### trainModel(self: Self, whichMold: int)

This function outlines the process of training a GaussianProcessRegressor model. Begins by checking if the data for whichMold has been changed since the last training session using dataChangedSinceLastTrain. If it hasn’t the function prints a message stating that no retraining is necessary and exits. Otherwise, the function uses \_\_extractConfig to build the correct configuration settings from config.json, then connects to the SQL database and pulls down the most recent lot data. If there are fewer than n\_minimum\_data\_points entries, training is aborted due to insufficient data. Otherwise, the rest of the config parameters are passed as arguments to the gaussian process kernel, which is built, compiled, and trained on the data. The resulting model has its data dumped to a file and uploaded to the database, and the index.json entry for whichMold is updated.

## Appendix D: train.py

#### Functions & Classes

##### trainOne(whichMold: int)

This function instantiates a GPFactory and calls the trainModel function for mold whichMold.

##### train()

This function enables logging, then instantiates a GPFactory object and a DBConnection object. The DBConnection object is used to get a list of all mold ids from the SQL database, then the GPFactory is used to train a model for each mold. Logging is toggled off once all training is completed.