PrismRCL - Technical Documentation - v2.7.2

I. Introduction:

For installation instructions and how to get started, please refer to the attached “Start Guide”. Please note that PrismRCL (starting with version 2.7.0) is a console application and is designed to run from the command line as a developer tool. It does not have a graphical user interface. All inputs are loaded from files on the file system. Similarly, all logs and outputs are directed to text files on the file system and are fully configurable by the user as shown in this document and in the “Start Guide”.

New in version 2.7.2: PrismRCLM now supports loading a model in SSL mode on all platforms. It has been successfully tested on Windows (x86_64 and aarch64), Linux (x86_64 and aarch64), and Darwin (x86_64 and aarch64).

New in version 2.7.1: PrismRCL now supports MacOS (Darwin) and ARM processors. It has been successfully tested on Windows (x86_64 and aarch64), Linux (x86_64 and aarch64), and Darwin (x86_64 and aarch64).

New in version 2.7.0: PrismRCL now supports Linux. It has been successfully tested on Ubuntu 22 & 24, as well as Red Hat Enterprise 9 and 10 and Fedora Workstation 42.

New in version 2.5.3: PrismRCL can now be run in a Docker container. It was tested on Windows containers. The containerized version is available only for Enterprise clients.

The Windows version contains two main executables: **PrismRCL.exe** and **PrismRCLM.exe**

The Linux/Darwin version contains two main executables: **prismrcl** and **prismrclm**

PrismRCL.exe (prismrcl) is a full-service application that can perform all supported tasks.

PrismRCLM.exe (prismrclm) is a web interface for PrismRCL (prismrcl) and is used primarily for loading pre-trained models and keeping them resident in memory so that they can be used for fast inference.

Each of the following parameters can be added following a command line execution of PrimsRCL.exe / prismrcl (or PrismRCLM.exe / prismrclm where applicable).

All parameters should be entered in lowercase characters. Supports .png images and .txt files. Text files can contain text or tabular data (numerical, text or mixed).

New in version 2.4.0: Tabular data no longer needs to be normalized.

Note: Models trained on previous versions of PrismRCL will not work with version 2.7.0 or later. Old models will have to be retrained.

For tabular data, please make sure that for each sample, the features are space-separated and stored in a single line text file. Normalization of numerical tabular data values is no longer required.

II. List of PrismRCL parameters and their functions:

auto-optimize

automatically sets evaluation method, rclticks, boxdown, directional, patternreduce, skipzero (defined below) using a "greedy validation algorithm with projection". auto-optimize uses the train portion of the data only, the resulting model can subsequently be tested with the test portion of the data. auto-optimize is a very efficient way of finding ideal parameters for new types of data not encountered before.

New in version 2.5.0: auto-optimize will automatically pursue different goals as follows:

Two classes: Overall Accuracy (acc)

If the dataset is reasonably balanced: Macro Average F1 Score (af1)

If the dataset is imbalanced: Weighted Average F1 Score (wf1)

Note: in addition to the functionality above, auto-optimize can also be forced to pursue the desired goal, by passing the auto-optimize parameter in this format:

auto-optimize=acc (auto-optimize will pursue Overall Accuracy)

auto-optimize=af1 (auto-optimize will pursue Macro Average F1 Score)

auto-optimize=wf1 (auto-optimize will pursue Weighted Average F1 Score)

auto-optimize=mcc (auto-optimize will pursue Matthews Correlation Coefficient)

Example 1 (Windows and Linux/Darwin):

saves the parameters to the log directory in **_optimize_summary_mm_dd_yy_hh_mm_ss.txt**.

*Note: an additional parameter, called **imaginaryslice** might also be generated and stored in the summary file. If it is, be sure to pass it along with the other parameters when you train your model.*

Note: we show how to create your Linux/Darwin command based on the Windows command in this first example only. The application works the same way on all platforms. On Windows, the paths follow the Windows operating system rules, and on Linux/Darwin, the paths follow the Linux/Darwin operating system rules.

Windows: `C:\PrismRCL\PrismRCL.exe` **`auto-optimize`**
`data=C:\data\train_data log=c:\log_files\`

Linux/Darwin: `/home/prismrcl/prismrcl` **`auto-optimize`**
`data=/home/data/train_data log=/home/log_files/`

Example 2 (Windows):

creates a model using parameters from auto-optimize (optimized parameters are stored in model).

`C:\PrismRCL\PrismRCL.exe` **`auto-optimize`** `data=C:\data\train_data savemodel=c:\models\best_model.model log=c:\log_files\`

Example 3 (Windows):

creates a model using parameters from auto-optimize (optimized parameters are stored in model) and evaluates model on test data, all in one pass.

`C:\PrismRCL\PrismRCL.exe` **`auto-optimize`** `data=C:\data\train_data testdata=C:\data\test_data savemodel=c:\models\best_model.model log=c:\log_files\`

New in version 2.6.0: PrismRCL now supports training text datasets for language modeling. We introduce a new parameter called "llm" which will instruct PrismRCL to treat the input dataset as a Large Language Model dataset.

nocache

bypasses caching when running auto-optimize – no impact on training sessions

Example (Windows):

creates a model using parameters from auto-optimize but do not cache the data (optimized parameters are stored in model).

`C:\PrismRCL\PrismRCL.exe auto-optimize` **`nocache`** `data=C:\data\train_data savemodel=c:\models\best_model.model log=c:\log_files\`

patiencesecs

sets the number of seconds to wait if nothing has improved in auto-optimize (default: 3600 seconds). Setting "patiencesecs=-1" will cause the system to wait infinitely.

useimaginary"

forces auto-optimize to also test the evaluation “fractal imaginary” (since the evaluation “fractal imaginary” is rare, it is better to have this be a conscious choice).

llm

Instructs PrismRCL to train the input text dataset for large language modeling.

The Random Contrast Learning (RCL) algorithm requires text data to be in a special format and structure before it can be used for LLM training. Data preparation starts with one file (or more) of plain text data without any formatting or markup (such as HTML, XML, JSON, etc..).

It is recommended to perform basic cleaning and filtering on the input text data prior to formatting and training, like removing very long words (possibly several words merged into one) and unwanted characters (like control characters as well as non-readable/non-printable characters).

Next, we use the text data to build what we refer to as input sequences and their target tokens using a sliding window mechanism. RCL requires that LLM data be formatted such that the input sequences are data samples that belong to a class (represented by the target token). Given a stream of text, we start at the beginning. We count the length of our sequence (e.g., 4 tokens), and the next token (e.g., fifth token) will be the target or class. We shift the input by one token to the right, then the next token will be the target or class. We repeat this step until the end of the input text stream.

Example text stream: The Project Gutenberg eBook of Les Misérables.

Input Sequence 1: The Project Gutenberg eBook -> Target Token 1: of
Input Sequence 2: Project Gutenberg eBook of -> Target Token 2: Les
Input Sequence 3: Gutenberg eBook of Les -> Target Token 3: Misérables
Input Sequence 4: eBook of Les Misérables -> Target Token 4: .

The data is now ready to be saved on disk having been properly formatted and prepared for RCL training.

We start by creating blank target token (class) folders. If a folder cannot be created by the operating system for whatever reason, its path information should be stored in a “failed” list to be examined and addressed later. If the operating system fails to create one or more folders, the RCL dataset creation should be interrupted to review and address the failures.

Once all class folders are created successfully, we then write all input sequences to a single text file (like “values.txt”; one sequence per line) in their corresponding class folders. The data is now ready for RCL training.

Note: Look for more detailed instructions on how to create RCL-ready LLM datasets on our website.

Usage (Windows):

```
C:\PrismRCL\PrismRCL.exe llm data=C:\llm-text-data\train testdata=
C:\llm-text-data\test
```

Note: The command above is incomplete. To learn how to train RCL on LLM text data, please refer to the included Start Guide for complete examples.

loadmodel

Specifies an existing model to be loaded for inference or to have other models added to it.

Example (Windows):

```
C:\PrismRCL\PrismRCL.exe loadmodel=c:\models\model012.model
testdata=c:\RCLC\data\test_data
```

Example (Windows):

```
C:\PrismRCL\PrismRCLM.exe loadmodel=c:\models\model012.model port=8080
```

Note: Using PrismRCLM.exe to load a model will start a web service that is accessible via HTTP and will keep the model resident in memory and ready for inference, without having to reload the model each time. In the example above, the instance will be accessible via http://localhost:8080/

Tip: Multiple models can be loaded on the same physical (or virtual) server using different port numbers. A model will be identified by the port number assigned to the web service when it was started.

The PrismRCLM web instance will respond with one of the following four statuses:

ready: if called with no parameters, and if server is idle.

done: when server has finished processing a request.

busy: if server is busy processing another request.

error: if request contains an error and server could not process it.

Running inference on the HTTP-loaded model is simple. Please see examples of web inference requests in section III of this document.

transferlearn

Specifies an existing model to be augmented by additional training on new data of similar shape as within the model. The settings for subsequent training will be set to the same as the transferlearn model. If the shape of the new data is different from the model specified as transferlearn the system will cast an error to the directory specified under the log parameter.

Example (Windows):

```
C:\PrismRCL\PrismRCL.exe transferlearn=c:\models\goodmodel.model
data= C:\data\moretrainingdata savemodel=c:\models\bettermodel.model
```

data

Specifies the folder for training data (and possibly test data). The ratio between the two can be set using testsize parameter (default testsize=0.1)

The data parameter can be applied stand-alone alone or in conjunction with the transferlearn parameter. If the data parameter is specified along with the loadmodel parameter. Example (Windows):

```
C:\PrismRCL\PrismRCL.exe data=c:\data\train_data
savemodel=c:\models\mymodel.model
```

From the folder specified as data the classes should be folders underneath with their respective .png or .txt files in each folder. *Please note that all file names of .png or .txt files must be unique across classes.*

addmodel

Specifies a model or set of models (separated by semicolon) to be added to an existing model. The add requires no additional subsequent training run to be functional at the accumulated models, thus the *addmodel requires a loadmodel to be in the same command*. If the training parameters for a loadmodel are different from the model specified as addmodel the system will cast an error to the directory specified under the log parameter. Individual models must also have been created using chunks from the same dataset.

Note: Models created with version 2.7.0 are not backwards compatible with previous versions of PrismRCL.

Examples (Windows):

```
C:\PrismRCL\PrismRCL.exe loadmodel=c:\models\bettermodel.model
addmodel=c:\models\model222.model
savemodel=c:\models\bettermodelx2.model
```

```
C:\PrismRCL\PrismRCL.exe loadmodel=c:\models\bettermodel.model
addmodel=c:\models\model111.model;c:\models\model222.model
savemodel=c:\models\bestmodel.model
```

savemodel

Specifies the resulting model for any of the above steps.

See example above.

testsize

As mentioned in the data section, testsize specifies the size of the split from the data folder to be served for testing of a trained model. If it is desired to test on a loadmodel (with subsequent addmodel) then use the testdata setting below. Default is testsize=0.1, but for example if a test of 20% of the data is desired, then pass the parameter testsize=0.2

testdata

This parameter specifies a folder dedicated to test data only. This is typically the parameter to use when inferring with an existing model.

Example (Windows):

```
C:\PrismRCL\PrismRCL.exe loadmodel=c:\models\model333.model
testdata=c:\data\test_data
```

testsave

use this parameter to save a dedicated test set that is equivalent to that used in memory as defined by testsize. This applies only to a run where there is training involved; otherwise, the folder referred should simply be passed as testdata (see above). The files in the resulting folder will be copies of their originals.

Example (Windows):

```
C:\PrismRCL\PrismRCL.exe loadmodel=c:\models\model444.model
testdata=c:\data\test_data testsave=c:\data\saveforlater
```

chisquared

Is a single parameter (no = value should be passed). This is an evaluation method.

It is also applied by default if no evaluation-type is passed.

"chisquared" (Chi-Squared) evaluates all patterns assuming the null hypothesis is true, i.e., all patterns are random. The class with the most "pull" away from random wins.

chisquaredpair

Is a single parameter (no = value should be passed). This is an evaluation method.

"chisquaredpair" works similarly to chisquared but it is for multi-class problems (3 classes or more), it looks for the best fit, pairwise, by evaluating two classes at a time.

fractal

Is a single parameter (no = value should be passed). This is an evaluation method.

"fractal" evaluates all patterns in accordance with their distance from a Probability Density Function (PDF) that assumes self-similarity (fractals) in the images. The class with the least Sum Squared Error (SSE) wins.

naivebayes

Is a single parameter (no = value should be passed). This is an evaluation method.

"naivebayes" (Naive Bayes) evaluates all patterns as independent with random as contrast for all other classes. The class with the most "pull" away from other classes wins.

naivebayespair

Is a single parameter (no = value should be passed). This is an evaluation method.

"naivebayespair" works similarly to naivebayes but it is for multi-class problems (3 classes or more), it looks for the best fit, pairwise, by evaluating two classes at a time.

softmax

Is a single parameter (no = value should be passed). This is an evaluation method.

“softmax” converts the model's raw output scores into a probability distribution, facilitating the evaluation of class probabilities in classification tasks. The predicted classes with the highest probabilities win.

softmaxpair

Is a single parameter (no = value should be passed). This is an evaluation method.

“softmaxpair” works similarly to softmax but it is for multi-class problems (3 classes or more), it looks for the best fit, pairwise, by evaluating two classes at a time.

imaginary

Is a single parameter (no = value should be passed). It is flag used with the **fractal** evaluation method.

“imaginary” can be applied to all evaluation methods.

When “imaginary” is passed along with the evaluation method, PrismRCL will collapse low frequency patterns into an imaginary pattern and use this as part of the given evaluation method selected. The effect of using “imaginary” will (depending on the data) result in smaller models with faster inference.

NOTE: that “imaginary” setting is lossy, and thus adding models with different settings for imaginary is not possible. “imaginary” may work well for certain types of data but can be too lossy for others.

trainacc

Is a single parameter (no = value should be passed). This setting forces PrismRCL to also output training accuracy. In general, it is desired only to use this, when necessary, since the training set can otherwise be freed from memory incrementally as PrismRCL trains.

Example (Windows):

```
C:\PrismRCL\PrismRCL.exe loadmodel=c:\models\model941.model
testdata=c:\RCLC\data\test_data trainacc testsave=c:\data\saveforlater
```

channelpick

Is for image training and inference only. channelpick allows PrismRCL to train on specific RGB channels of an image or either average or a combination as follows:

1: red
2: green

3: blue
4: average of red, green, and blue (also default for monochrome images)
5: combining all (default setting)

readtextbyline

Is for text training and inference. It indicates that each text file encountered may contain one or more training samples, and that each line of text should be considered a sample. If this setting is not specified, PrismRCL will treat the contents of each input text file as a single data sample. This parameter is required for RCL LLM datasets that are prepared using the compact text data format (recommended), meaning that a single text file contains multiple data samples, one sample per line.

rclticks

Is for image and text training and inference. It is a setting for how granular PrismRCL should interpret a value from channelpick. The default setting is 10, meaning that PrismRCL discretizes a byte into round (255/10) values.

directional

directional instructs the algorithm to maintain the order of the features in a data sample. This applies to text and image data. It has no effect on tabular data.

patternreduce

patternreduce refers to the threshold a pattern must clear to be taken into account by the evaluation method. The default setting for this parameter is 0.6 and can be increased to 0.95. note: the best setting for this parameter is usually discovered by auto-optimize.

skipzero

This parameter works only for image data. It is applied if the training images are black/white images detected or when channelpick (see definition above) is set to 4. If set, the algorithm will ignore all zero RGB values.

sectors

This parameter works only for image data. The default value is 0. As “sectors” increases, the algorithm will interpret the image as multiple regions or tiles, depending on the set value (1,2,3,..).

filter

Is the size of PrismRCL's sliding window through which an image is observed. Default setting is filter=3

stride

Stride is the movement of the filter over an image. Default setting is stride=1

Interpretation of the filter:

PrismRCL offers two different approaches to interpreting what is "seen" in the filter: **cluster** and **directional**.

"cluster" is robust to angle and direction of an image. (This is the default setting)

"directional" is more detailed and interprets direction.

boxdown for image data:

PrimsRCL offers a style of folding an image into a smaller area over which the filter is passed: boxdown. boxdown does a pixel average. By default, this technique is not applied. To activate it, use boxdown=1 (to fold once). Any integer can be applied, but obviously the size of an image puts a limitation on how many folds make sense. This can also be discovered through auto-optimization.

boxdown for text/tabular data:

PrimsRCL offers the same algorithmic parameter for text and tabular data: boxdown. By default, this technique is not applied. To activate it, use the boxdown value obtained from your auto-optimization session.

Example of the settings (Windows):

```
C:\PrismRCL\PrismRCL.exe data=c:\data\train_data boxdown=0 filter=3
rclticks=20 saveif=0.9 stride=1 testsize=0.1
```

(The settings for down, filter, stride, and testsize are defaults; but written out here for illustration)

inftoimage (image data only)

This inference style lets you pick a model and a folder for testdata, then it classifies the images from the testdata folder and organizes them in accordance with classification. In addition, it augments the

classified images to grayscale while leaving only the pixels that are the algorithm's "reasons" for suggesting the image belongs to a particular class colored in red.

Example (Windows):

```
C:\PrismRCL\PrismRCL.exe loadmodel=c:\models\bestmodel.model
testdata=c:\data\test_data inftoimage=c:\results\images_out
```

tracecolor (image data only)

This flag allows you to set the color for the image overlay when you use inftoimage inference. It can be set to either: r,red,g,green,b,blue.

Example (Windows):

```
C:\PrismRCL\PrismRCL.exe loadmodel=c:\models\bestmodel.model
testdata=c:\data\test_data tracecolor=red
inftoimage=c:\results\images_out
```

inftotext (image data only)

This inference style lets you pick a model and a folder for testdata, then it classifies the images from the testdata and provides a list of the classes per file at the location identified by inftotext.

Example (Windows):

```
C:\PrismRCL\PrismRCL.exe loadmodel=c:\models\bestmodel.model
testdata=c:\data\test_data inftotext=c:\results\output.txt
```

textinftotext (text data only)

This inference style lets you pick a model and a folder for testdata, then it classifies the text files from the testdata and provides a list of the classes per file at the location identified by textinftotext.

Example (Windows):

```
C:\PrismRCL\PrismRCL.exe chisquared
loadmodel=c:\models\textmodel.model testdata=C:\data\chat_text_test
textinftotext=c:\results\test_out.txt
```

trainthreads

trainthreads apply the resources needed under training a model (the thread count specifically).

This is a setting that gives the ability to throttle the training machine from running out of resources. It is generally recommended (based on experiments across 4 types of hardware configurations) to set trainthreads to the be equal to the number of logical processors the machine has. Default is trainthreads=20

infthreads

infthreads apply the resources needed during inference with a model (the thread count specifically).

This is a setting that gives the ability to throttle the training machine from running out of resources. Since inference generally is way less computationally expensive, this number can be (much) higher than the trainthreads setting. Based on experiments across 4 types of hardware configurations we recommend setting this to either infthreads=180 (non-LLM) or infthreads=60 (LLM).

It is generally recommended (based on experiments across 4 types of hardware configurations) to set trainthreads to the be equal to the number of logical processors the machine has. Default is infthreads=1000

maxcpu (deprecated)

Note: This setting will be removed from future versions. Please use trainthreads and infthreads instead.

Is another “handle” to controlling the physical machine from running out of resources. This regulates how high the CPU usage can be (in percentage) before PrismRCL beings to “throttle down”.

The default setting is maxcpu=100

maxmem

Is the last handle that can be used to control PrismRCL’s behavior on the physical machine. Perhaps the most critical because running out of memory and thereby initiating memory/disk swapping has the worst throughput penalty of all.

Based on experiments on a 64 GB machine has led to setting this default to maxmem=30

New in version 2.5.0: maxmem is now set to 0.9 x system_memory by default.

Note: for auto-optimize sessions, the application will exit if the system runs out of memory. For training sessions, the application may not exit gracefully if it runs out of memory.

log

This sets the location of the logfiles that are generated during certain checkpoints to monitor progress of PrismRCL. Default setting is log= c:\temp\logfiles

temp

This sets the location of the temporary files writing during saving of data and models.

Default setting is temp= c:\temp

outputfile

This setting is optional and typically used for benchmarking. This setting gives a file-location where a file will summarize the following: timestamp, settings, total training time (secs), total inference time (secs), inference time per image (secs), as well as the test accuracy with untrained data.

Combined with this the parameter **saveif** can be used to control output only if test/validation accuracy is greater than this setting.

Example:

```
C:\PrismRCL\PrismRCL.exe data=c:\data\train_data saveif=0.9
rclticks=20 outputfile=c:\RCLC\results\output.txt
```

If the value for is set, PrismRCL will automatically shut down so multiple runs can be scripted in sequence.

stopwhendone

in case it is desired to shut PrismRCL down automatically after a run, passing stopwhendone will do this.

Example:

```
C:\PrismRCL\PrismRCL.exe loadmodel=c:\models\bettermodel.model
addmodel=c:\models\model945.model;c:\models\model555.model
savemodel=c:\models\bestmodel.model stopwhendone
```

promptest

This setting prompts the user before running inference, on whether to run (yes) or not to run (no).

This setting is useful for demonstrating PrismRCL as well as when measuring inference speeds.

silenttest

This setting will limit test output to confusion matrix only. “silenttest” is useful for demonstrating PrismRCL.

III. Using PrismRCLM for fast inference:

To run inference using PrismRCLM, a web request is made on the running instance of choice. The port number will indicate the model to be used for inference. In addition, the evaluation type, the input data, and the prediction output must be specified. You are encouraged to generate log files for your inference sessions, although this is not required. The following are some examples.

Inference to text on an image data model:

http://locahost:8080/evaluation_method&imaginary_flag&testdata=C:/RCLC/data/test_data/&log=C:/RCLC/logfiles/&inftotext=C:/RCLC/output/prediction_output.txt

IMPORTANT: Paths specified in the request URL must match the operating system paths and must be accessible locally on the system where the PrismRCLM instance is running (network drives/shares are supported).

port number: indicates the port number where your model is loaded. Multiple models can be identified by their port numbers.

evaluation_method: chisquared | fractal | naivebayes | softmax

Note: Evaluation method must match the method used for training the model (does not need to be passed explicitly, since it is stored in the model).

imaginary_flag: &imaginary if enabled, blank otherwise.

Note: The PrismRCL parameter is “imaginary”. You may choose to hardcode it in the URL if you prefer: “**&imaginary**”

testdata: path to folder containing samples to run inference on.

log: path to folder where log files for the inference session will be stored.

inftotext: path to file where predictions will be stored. The file will contain a list of inference samples and their predicted classes.

Inference to image on an image data model:

http://locahost:8080/evaluation_method&imaginary_flag&testdata=C:/RCLC/data/test_data/&log=C:/RCLC/logfiles/&inftoimage=C:/RCLC/output/predictions/

inftoimage: path to folder where augmented image predictions are stored (see parameter explanation in section 2 for more details.)

Inference to text on a text or tabular data model:

http://locahost:8080/evaluation_method&imaginary_flag&testdata=C:/RCLC/data/test_data/&log=C:/RCLC/logfiles/&textinftotext=C:/RCLC/output/prediction_output.txt

textinftotext: path to file where predictions will be stored. File will contain a list of text or tabular inference samples and their predicted classes.

IV. Starting PrismRCLM in SSL Mode:

Starting with version 2.7.2, PrismRCLM can be started in SSL mode on all platforms. Below are the instructions for all supported builds.

1. **For Windows Operating Systems:**

   Our current release supports SSL using SChannel on Windows. When using SChannel, a .PFX certificate is required, and OpenSSL libraries are not needed during normal operation. OpenSSL is only necessary if a .PFX file is not already available, in which case OpenSSL is used to generate the .PFX certificate.

   Generating a .PFX file:

   - Install OpenSSL for Windows from **https://slproweb.com/products/Win32OpenSSL.html**

     (a “Light” version is sufficient).

- If serverkey.pem and servercert.pem are not available, generate them from PowerShell with the following command (self-signed):
    - **"C:\Program Files\OpenSSL-Win64\bin\openssl.exe" req -x509 -newkey rsa:4096 -keyout serverkey.pem -out servercert.pem -days 365 -nodes**
- With serverkey.pem and servercert.pem available, generate server.pfx with:
    - **"C:\Program Files\OpenSSL-Win64\bin\openssl.exe" pkcs12 -export -inkey serverkey.pem -in servercert.pem -out server.pfx -name "MyServerCert"**

Notes:

- Do not create a password for the .pfx file. Press ENTER to skip generating a password.
- Self-signed certificates created with the above commands are intended for development and testing only. For production environments, obtain certificates from a recognized certificate authority.

Starting PrismRCLM in SSL Mode on Windows:

```
C:\PrismRCL\PrismRCLM.exe loadmodel=c:\path\to\model\model_name.model
port=443 pfxfile=C:\PrismRCL\server.pfx log=\path\to\logfolder\
```

This will load a secure instance at https://server.mydomain.com/

Note: Your SSL files must match the domain name of the host where you will be starting the secure web instance.

2. **For non-Windows Operating Systems:**

Assuming the SSL certificate and key files have been generated or obtained, PrismRCLM can be started on non-Windows using the following command syntax:

```
/prismrcl/prismrclm loadmodel=/path/to/model/model_name.model port=443
certificate=/prismrcl/domaincert.pem key=domainkey.pem temp=/temp
log=/path/to/logfolder/
```

This will load a secure instance at https://server.mydomain.com/

Notes:

- All necessary SSL libraries are provided with our software release.
- Once the model is loaded, inference can be run using HTTPS like the non-SSL examples provided.