A. Attachments

A.1 User manual for the developed tools

This chapter is dedicated to the description of the implemented tools for the clustering, proposed in the thesis above. It consists of three programs:

- *clusterer*,
- *window_processor* and
- *window_trigger_creator*.

Each of the programs has a dedicated section where we discuss the installation and usage of the program.

Currently, for the clusterer and window_processor programs, there are two options for installation – either to build it from the source, or to download its pre-built executable. For the window_trigger_creator, we provide only the build option. Distribution via a single executable would need to contain all dependencies, including large machine-learning libraries, which would take a lot of space (more than a gigabyte).

- Build from source:
  - `https://github.com/TomSpeedy/clusterer`
  - `https://github.com/TomSpeedy/window_processor`
  - `https://github.com/TomSpeedy/window_trigger_creator`

  Properties:

  + Wider support of various operating systems.
  + Possibility of extending the source code.
  − Requires more time to set up dependencies.

- Download pre-built executables (`https://drive.google.com/drive/folders/1DOCHbyQ9w5nK9EDZub-YpGhHROj_7BhL?usp=sharing`).

  Properties:

  + Fast and simple.
  − Currently only supports Ubuntu OS and 64-bit Windows.

For the purpose of testing all three applications, we share a (relatively) small dataset, see the link `https://drive.google.com/drive/folders/1DOCHbyQ9w5nK9EDZub-YpGhHROj_7BhL?usp=sharing`. In case of any problems or questions, please send an email to 'celko.tom@gmail.com'.

A.1.1 Clusterer

The clusterer is a command-line tool meant for the first step of event building called clustering. As of now, the clusterer works in the offline mode, processing hits obtained from Timepix3 in a text file format.

Installation

- From pre-built binary – download the zipped folder and simply run the executable. (named 'clusterer' or 'clusterer.exe')
- Build from source:

  Prerequisites:

  1. OnnxRuntime (version for Ubuntu is distributed within the clusterer repository).
  2. Cmake version $\geq$ 2.8.
  3. C++ compiler with support of C++17 standard. (GCC version 8+, Visual Studio 2017 15.8 – MSVC 19.15+ or similar).

  Note: Steps marked by * are not required for Linux systems.

  1. Clone the repository using HTTPS or SSH.
  2. *Download OnnxRuntime from `https://onnxruntime.ai/`. Select 'optimize for inference' and choose the target platform. Choose 'C++' API and in 'hw acceleration' field choose 'default cpu'. Follow the instructions for installation.
  3. Create the build directory ('clusterer/build/') and navigate to it.
  4. Run **`cmake ..`**, this generates the build files. For the MSVC compiler, this generates VS solution. For this solution, the Onnx runtime needs to be added in the form of a NuGet package.
  5. *Set environmental variable'ONNX_LIBRARY_PATH' to the directory where the onnxruntime library is located (search for a file named 'onnxruntime.dll', 'onnxruntime.so' or similar).

6. Run **`cmake --build .`**. For the MSVC it is required to add **`--config Release`** as the default build is Debug. The executable is then located in the 'Release' or 'bin' folder. If the build does not succeed, observe the error message, it might point toward the problem. Alternatively, contact the author.

Usage

In order to use the clusterer, the user is expected to provide command line arguments. To display the options pass the **`--help`** argument. The arguments and the options are always separated by a blank space. The clusterer can be run in multiple configurations, specified by the 'mode' option.

- `benchmark` – To run this option, please download the test dataset from the link `https://drive.google.com/drive/folders/1DOCHbyQ9w5nK9EDZub-YpGhHROj_7BhL?usp=sharing` and extract it. This option requires only a single additional argument – path to the downloaded dataset.
- `window_processor` – This option serves to generate file with unclustered window features, further used in the window processor app. Additionally, the calibration folder should be specified using `--calib` option.
- `compare` – The 'compare option is a tool for comparison of two clustered files. It computes the Intersection over union match between the cluster datasets.
- `clustering` – The default mode, which clusters the given data file. Similarly to window_processor, the calibration folder should be provided.

The arguments for the program are passed in two ways. Some of the arguments are passed directly via the command line. Additional 'Node arguments' for the computational nodes are passed in a file via `--args` argument. All options and arguments:

- `--mode` <one of 'clustering', 'window_features', 'compare', 'benchmark'>
  – defaults to 'clustering'
  – mandatory argument if 'clustering' or 'window_features': `--calib`
- `--help`
- `--args` <path_to_node_argument_file>– a file with a set of parameters for each node in the computational graph
- `--calib` <path_to_calibration_folder>

- `--merge_type` <one of ‘none’, ‘simple’, ‘single_layer’, ‘tree’ or ‘multioutput’>
  – defaults to ‘single_layer’ – chooses the type of merging the datastreams, if ‘none’ is chosen, the no data-based split takes place, and ‘n_workers’ parameter is ignored

- `--output` <path_to_output_folder>
  – a directory where the output is written
  – defaults to <binary_location>/../../output

- `--debug`
  – prints extra information about the dataflow in the computational graph, which can be useful for debugging purposes

- `--n_workers` <positive integer>
  – the width of the data-based split for the clustering architecture
  – defaults to 4

- `--clustering_type` <one of ‘standard’, ‘temporal’, ‘tiled’, ‘halo_bb’>
  – defaults to ‘standard’

Node argument file

In ‘node arguments’ configuration file, the user can override the default parameters of each node by specifying the `--args` argument file. The file has the following structure:

```
node_name
–[node_property1_name]:[node_property1_value]
–[node_property2_name]:[node_property2_value]
another_node_name
...
```

Currently, the following node names and properties are supported:

- reader
  - split:[‘true’ or ‘false’]
    Sets if the reader node should perform the split for parallel clustering computation if ‘merge_type’ is ‘none’. The first node (in the order of the data flow) with this flag set to ‘true’ is the split node. The default is ‘true’.
  - sleep_duration_full_memory:[positive real] Sets the duration for which the reader is inactive after the memory limit is reached (in microsec-

onds). After this time, if the memory check succeeds, the processing is restored. Defaults to 100.

- calibrator
  - split:[‘true’ or ‘false’]
    Analogically, see the same option for reader node.
- sorter
  - split:[‘true’ or ‘false’]
    Analogically, see the same option for reader node.
- clusterer
  - max_dt:[positive real]
    The maximum time in nanoseconds for hits to be considered ‘temporally neighboring’. During clustering, two hits A and B belong to a single cluster if there exists a path of hits where each hit on the path is spatially and temporally neighboring. It defaults to 200.
  - tile_size:[one of 1, 2, 4, 8, 16]
    If set to a value $d > 1$, the detector is split into tiles of size $d \times d$ pixels. When checking the spatial 8-neighborhood of a pixel, all pixels in the neighborhood of a tile are considered as neighboring to this pixel. This can effectively ignore the ‘dead’ pixels. The default is 1.
- trigger
  - use_trigger:[‘true’ or ‘false’]
    An important variable, indicating if the trigger should be used. The default is ‘false’.
  - window_size:[positive real]
    The size of a time window (in nanoseconds) after which the window statistics are evaluated. **Please, use the same value which was used by the ‘window computer’ node.** Use with care; a small value of this parameter can significantly slow down the processing. The default is $2 \cdot 10^8$ ns (200ms).
  - diff_window_size:[positive real]
    The size of a time window (in nanoseconds) with respect to which the features are differentiated. After that, the median of these differences is chosen as the difference value. A value smaller than window_size implies no differentiation. Too high a value can notably slow down

  the processing. **Please, use the same value which was used by the 'window computer' node.** The default is $1 \cdot 10^9$ ns (1 s), which corresponds to five default time windows.

  - trigger_time:[positive real]
    The trigger time is the time for which the trigger remains active after the initial activation. If the trigger reactivates during this time window, the active time of the trigger is extended adequately. The default is $5 \cdot 10^8$ (500 ms).
  - trigger_file:[a path to a trigger file]
    The trigger file path should point to the existing trigger file created by the 'window trigger creator' application. This trigger file is a (possibly machine-learning) model serialized in an ONNX format. The suffix is used to determine the correct type of the model – please do not modify (.nnt = neural network (from tensorflow package), .svmt = support vector machine, .osvmt = one-class support vector machine (from scikit-learn package)). The type of model needs to be assessed correctly, as some of the models produce multiple outputs of various data types. Alternatively, if the suffix is '.ift', the trigger file contains serialized intervals of features when the trigger should be active.

- halo_bb_clusterer
  - window_size:[positive real]
    The time window for hits that are clustered in the same buffer.
- window_computer
- window_size:[positive real]
  Analogical to the same property of the trigger node. It can be set via the 'window computer' GUI. The default is $2 \cdot 10^8$ (200ms).
- diff_window_size:[positive real]
  Analogical to the same property of the trigger node. It can be set via the 'window computer' GUI. The default is $1 \cdot 10^9$ (1s).

Example use cases:

- `./clusterer --calib [path/to/my/calib/folder/] [path/to/file/for/clustering]`
- `./clusterer --calib [path/to/my/calib/folder/] --args [path/to/node_args_file.txt] [path/to/file/for/clustering.txt]`

- `./clusterer --mode benchmark [/path/to/clusterer_data]`
- `./clusterer --mode compare [/path/to/first_clustered_file.ini] [/path/to/second_clustered_file.ini]`

A.1.2 Window processor

Installation

- From pre-built binary – download the zipped folder and simply run the executable. (named 'window_processor' or 'window_processor.exe')
- Build from source:

  Prerequisites:

  1. OnnxRuntime (version for Ubuntu is distributed within the clusterer repository).
  2. Cmake version $\geq$ 2.8.
  3. C++ compiler with support of C++17 standard. (GCC version 8+, Visual Studio 2017 15.8+ – MSVC 19.15+ or similar).
  4. Build clusterer from the source. Apart from the executable, the build generates a clusterer library (.so,.dll,...), required by the window processor.

  Note: Steps marked by * are not required for Linux systems. After the prerequisites are matched, you may proceed further.

  1. Either place the window_processor project in the same directory as the clusterer, or preferably, set the environmental variable 'CLUSTERER_PATH' to the parent folder of the clusterer project.
  2. Proceed similarily to the build procedure of clusterer (as shown below).
  3. Create the build directory ('window_processor/build/') and navigate to it.
  4. Run **`cmake ..`**, this generates the build files. For the MSVC compiler, this generates VS solution. For this solution, the Onnx runtime needs to be added in the form of a NuGet package.
  5. *Set environmental variable 'ONNX_LIBRARY_PATH' to the directory where onnxruntime library is located (search for a file named 'onnxruntime.dll', 'onnxruntime.so' or similar).

6. Run **`cmake --build ..`** For the MSVC it is required to add **`--config Release`** as the default build is ‘Debug’. The executable is then located in the ‘Release’ or ’bin’ folder. If the build does not succeed, check the error message, it might point toward the problem. Alternatively, contact the author.

Usage

1. Select the input file. You can either:

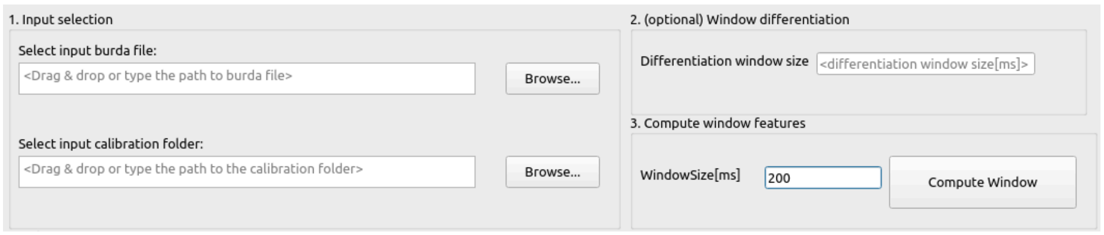
   - type the path manually,
   - click ‘Browse’ and navigate to the file, or
   - **drag and drop** the file into the text field.

   The input file should be in the Burda-file format, where each line corresponds to a hit and consists of four integers: linear coordinate of the pixel, time of the arrival using the low-frequency clock, time of arrival using the high-frequency clock, and time over the threshold.

2. Select the calibration folder. The same three options are available as mentioned above.

3. Optionally set the differentiation window size **in milliseconds**.

4. Set the window size, also **in milliseconds**.



1. Input selection

Select input burda file:

<Drag & drop or type the path to burda file>

Browse...

Select input calibration folder:

<Drag & drop or type the path to the calibration folder>

Browse...

2. (optional) Window differentiation

Differentiation window size <differentiation window size[ms]>

3. Compute window features

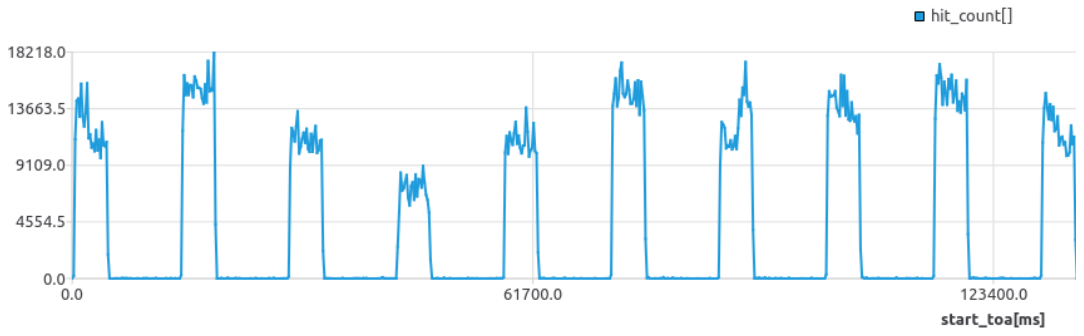
WindowSize[ms] 200

Compute Window

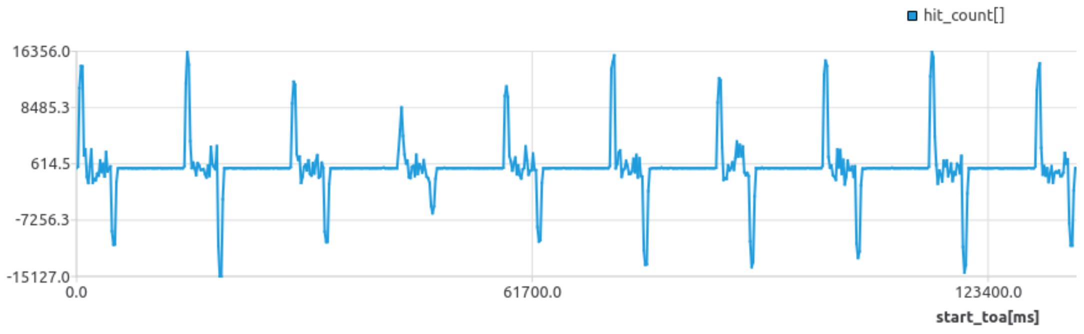
Figure A.1: Input selection fields, as described by the fields, as described by steps above.

5. Click ‘Compute window’. The table with windows and its plots should be displayed.

6. Click on the column in the header of the table to display the attribute’s plot with respect to the time below. Select the region on the x-axis to zoom in on the plot. Right-click on the plot to zoom out.



(a) Plot of non-differentiated feature (hit count) with respect to time.



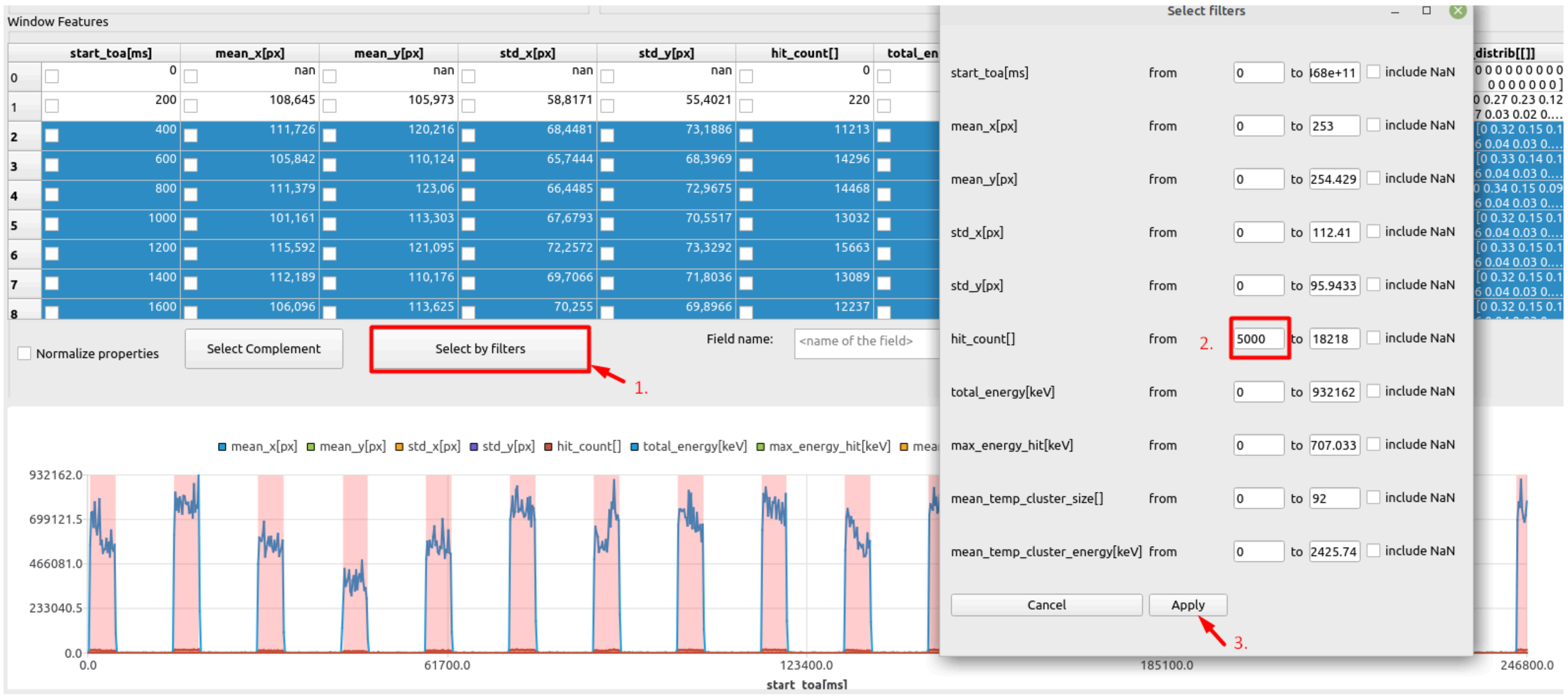
(b) Plot of differentiated feature (hit count) with respect to time.

Figure A.2: Comparison of the standard (not differentiated) features and their differentiated version.

7. Select windows of interest by hand by clicking on the leftmost field in the row (part of the table header). Keys like 'Ctrl' (to extend the selection) or 'Shift' (for interval selection) can be used in the table to select multiple windows.

8. Click 'Select by filters' and choose the interval for each property. Click 'Apply'. Only windows with properties within these intervals are **added to the selection (currently selected rows are NOT unselected)**. This allows you to use this button multiple times to combine multiple interval criteria by disjunction.

9. The selected windows are highlighted in the plot below.



Window Features

	start_toa[ms]	mean_x[px]	mean_y[px]	std_x[px]	std_y[px]	hit_count[]	total_en	distrib[[]]
0	0	nan	nan	nan	nan	0		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
1	200	108,645	105,973	58,8171	55,4021	220		0 0.27 0.23 0.12 7 0.03 0.02 0....
2	400	111,726	120,216	68,4481	73,1886	11213		[0 0.32 0.15 0.1 6 0.04 0.03 0....
3	600	105,842	110,124	65,7444	68,3969	14296		[0 0.33 0.14 0.1 6 0.04 0.03 0....
4	800	111,379	123,06	66,4485	72,9675	14468		0 0.34 0.15 0.09 6 0.04 0.03 0....
5	1000	101,161	113,303	67,6793	70,5517	13032		[0 0.32 0.15 0.1 6 0.04 0.03 0....
6	1200	115,592	121,095	72,2572	73,3292	15663		[0 0.33 0.15 0.1 6 0.04 0.03 0....
7	1400	112,189	110,176	69,7066	71,8036	13089		[0 0.32 0.15 0.1 6 0.04 0.03 0....
8	1600	106,096	113,625	70,255	69,8966	12237		[0 0.32 0.15 0.1

Normalize properties

Select Complement

Select by filters

1.

Field name: <name of the field>

Select filters

start_toa[ms]	from	0	to	}68e+11	include NaN
mean_x[px]	from	0	to	253	include NaN
mean_y[px]	from	0	to	254.429	include NaN
std_x[px]	from	0	to	112.41	include NaN
std_y[px]	from	0	to	95.9433	include NaN
hit_count[]	from 2.	5000	to	18218	include NaN
total_energy[keV]	from	0	to	932162	include NaN
max_energy_hit[keV]	from	0	to	707.033	include NaN
mean_temp_cluster_size[]	from	0	to	92	include NaN
mean_temp_cluster_energy[keV]	from	0	to	2425.74	include NaN

Cancel

Apply

3.

🟦 mean_x[px] 🟩 mean_y[px] 🟧 std_x[px] 🟪 std_y[px] 🟥 hit_count[] 🟦 total_energy[keV] 🟩 max_energy_hit[keV] 🟧 mea
932162.0
699121.5
466081.0
233040.5
0.0
0.0
61700.0
123400.0
185100.0
246800.0
start_toa[ms]

Figure A.3: Window selection, as described by the fields above. The boxes highlighted in red correspond to the steps described in the text.

10. Type in the name for the currently selected rows.

11. Click 'Save selected' to save the selected windows to the file.

12. If you want to use supervised learning for the trigger, return to the step with window selection. Select different windows (or possibly compute features of a different burda-file), select the class name for the selected windows, and **save it to the same file**. It will NOT completely rewrite the data; it will only **append** your selection to the file.

13. If an unsupervised learning method is to be used, all of the windows can be selected.

14. If a non-ML method is to be used, you still need to create an arbitrary data file selection (the selection can be empty).

15. After multiple iterations, the trigger data file is created and can be used by the 'window trigger creator' program.

A.1.3 Window trigger creator

The window creator app serves as a tool to create trigger files, further used by the clusterer program. On input, it expects a file with window features created by the window computer.

Installation

As the program itself has many dependencies from large packages, we decided against building the file into a large executable. Instead, we provide a simple alternative for installation.

Prerequisites:

- Python3 is installed and added to the system path.
- Python package manager 'pip' or 'pip3' is installed and added to the system path.

1. Clone the repository at `https://github.com/TomSpeedy/window_trigger_creator` using HTTPS or SSH.

2. The repository contains 'requirements.txt' file where required dependencies are listed. For installation of dependencies, navigate to the 'window_trigger_creator' directory and run in terminal **'pip install -r requirements.txt'**.

3. Check the command-line output and verify that the requirements were successfully installed. In case of an error, you can contact the author.

4. Navigate to the 'run_script' directory.

5. For Windows OS, navigate to the 'windows' directory. For Linux, navigate to the 'linux' directory.

6. To check the installation, double click on the 'window_trigger_creator.bat' or 'window_trigger_creator.sh' – the GUI should appear.

Usage

1. Load the file with window feature values (*.wf). You can again type it by hand, use the 'Browse' button or 'Drag and drop' the file into the text field. Then, click 'Load selected file', see Figure A.4.

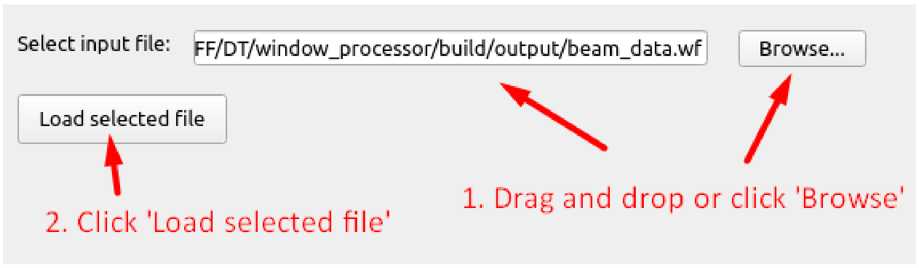
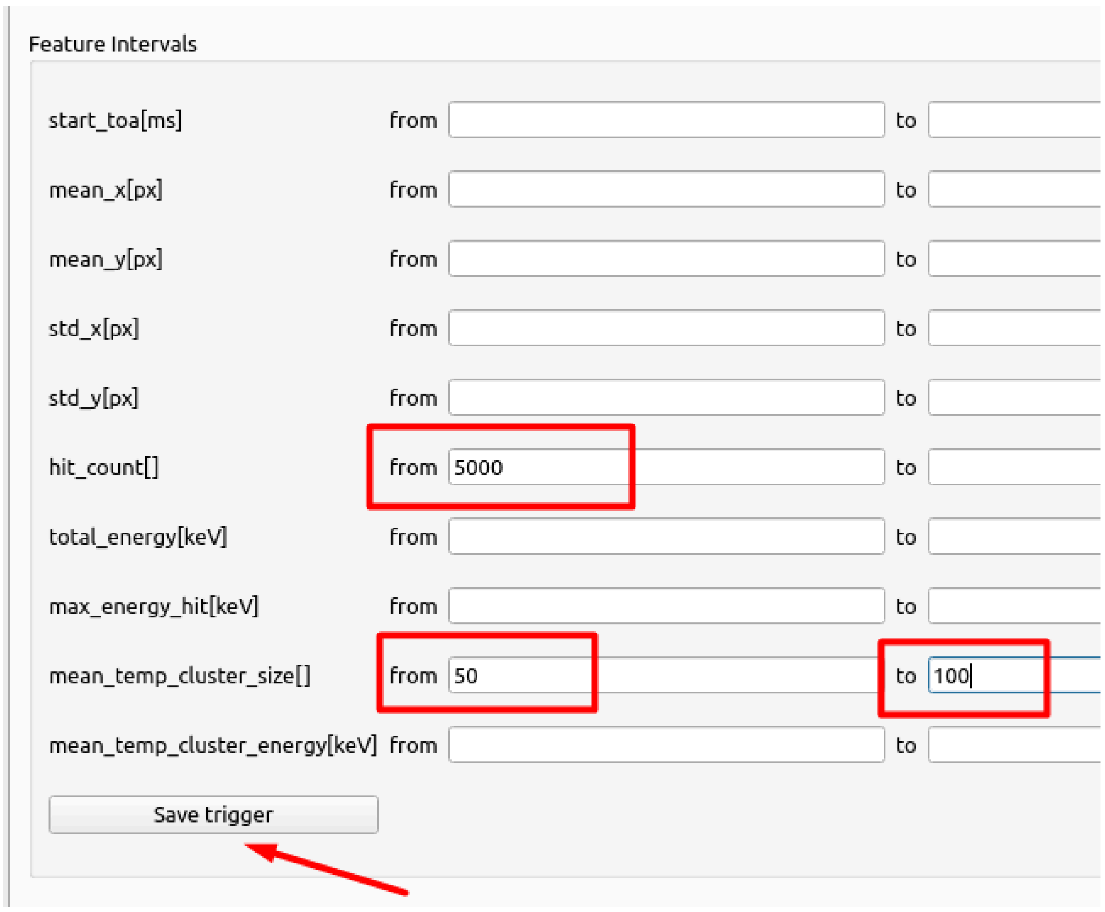

Select input file:
FF/DT/window_processor/build/output/beam_data.wf
Browse...
Load selected file
1. Drag and drop or click 'Browse'
2. Click 'Load selected file'

Figure A.4: Loading the window feature file.

2. After the file is loaded, choose the trigger type.

   - Manual interval trigger – choose the intervals of interest and click 'Save trigger'. To combine multiple triggers by disjunction, repeat this process and save the triggers to the same file see Figure A.5. The file will NOT be overwritten, it will only append the interval selection to the file.


Feature Intervals

start_toa[ms]	from		to	
mean_x[px]	from		to	
mean_y[px]	from		to	
std_x[px]	from		to	
std_y[px]	from		to	
hit_count[]	from	5000	to	
total_energy[keV]	from		to	
max_energy_hit[keV]	from		to	
mean_temp_cluster_size[]	from	50	to	100
mean_temp_cluster_energy[keV]	from		to	

Figure A.5: Creating the interval trigger.

- Supervised machine-learned trigger (SVM, NN), see Figure A.6.
  (a) Optionally, import hyperparameter file (again, you can use the 'Browse' button or 'Drag and drop'). The file is in JSON format, and the parameters are listed in Table A.1 and A.2.
  (b) Choose the classes which should trigger the clustering.
  (c) Click 'Start Training' and wait until the training is done. (It is shown in the log on the right.)
  (d) Click 'Save trained model'.

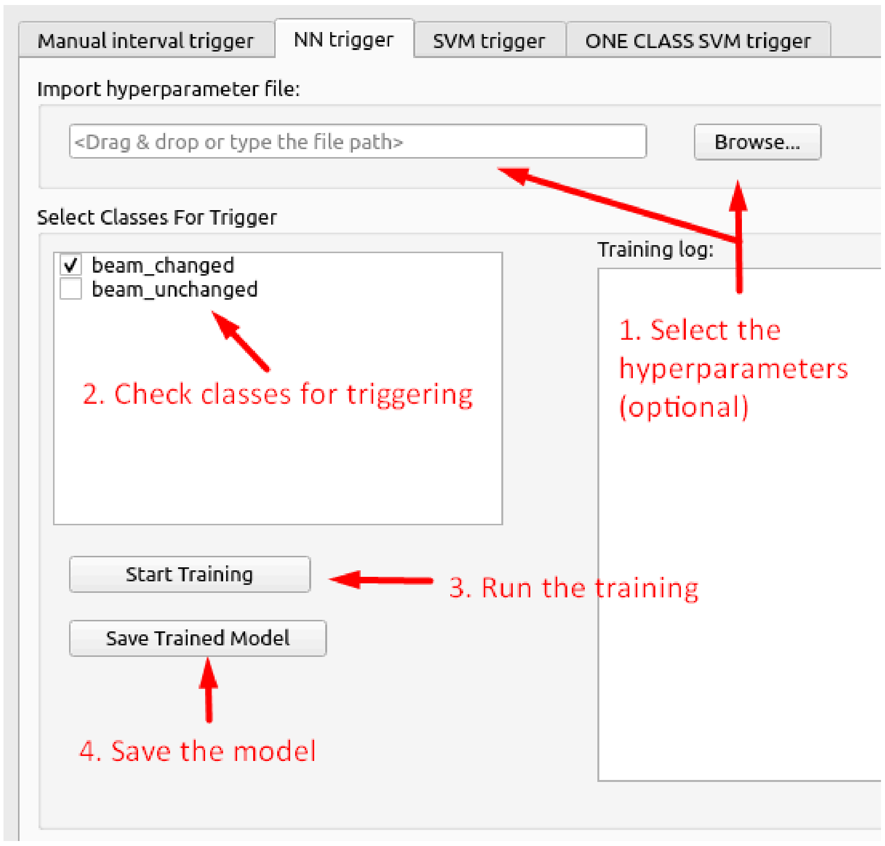


Figure A.6: Training supervised trigger.

- Unsupervised machine-learned trigger (one class SVM), see Figure A.7.
  - (a) Optionally, import hyperparameter file (again, you can use the 'Browse' button or 'Drag and drop'). The file is in JSON format, and the parameters are the same as in Table A.2, **except for the 'dataSplit' parameter**, which is left out (no splitting is taking place).
  - (b) Click 'Start Training' and wait until the training is done. (It is shown in the log on the right.)
  - (c) Click 'Save trained model'.

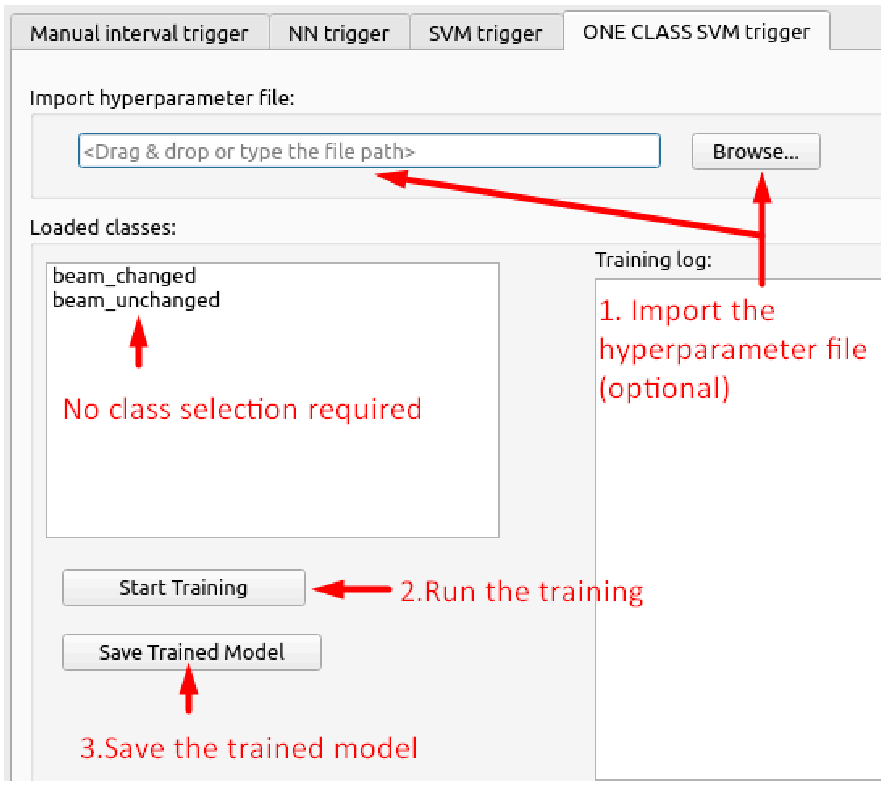


Figure A.7: Training an unsupervised trigger.

After the models are saved, they can be passed to the clusterer within the ‘node args file’ (‘--args’ option) as an argument ‘trigger_file’ to the ‘trigger’ node.

MLP hyperparameters		
Name	Type	Additional constraints
learningRate	floating point	It has positive value, recommended between 10^{-6} and 10^{-2}. The default is 10^{-3}
layerSizes	list of integers	It is a non-empty list of positive integers. The default is $[20, 20]$
learningRateDecay	dictionary	It contains fields 'name', 'initialLearningRate', and if 'name' is 'exponential' also 'decayRate'. It is optional, it has no default (no decay).
learningRateDecay .name	string	Is one of 'exponential' or 'cosine'.
learningRateDecay .initialLearningRate	floating point	It has positive value, recommended between 10^{-6} and 10^{-2}.
learningRateDecay .decayRate	floating point	It has positive value smaller than 1.
optimizer	string	Currently only 'Adam' is supported.
batchSize	integer	It has a positive value, the default is 5.
epochCount	integer	It has a positive value, the default is 30.
dataSplit	dictionary	It contains three fields 'training' 'test' and 'validation', each floating point number between 0 and 1, summing to 1. The default is 0.7 for training, 0.15 for validation, and 0.15 for testing.

Table A.1: Allowed hyperparameters of the multilayered perceptron and their domains.

SVM hyperparameters		
Name	Type	Additional constraints
kernel	string	It is one of 'rbf', 'linear', 'poly', 'sigmoid'. The default is 'rbf'.
gamma	string	It is one of 'scale' or 'auto'. The default is 'auto'.
dataSplit	dictionary	It contains three fields, 'training' and 'test', each floating point number between 0 and 1, summing to 1. The default is 0.8 for training and 0.2 for testing.

Table A.2: Allowed hyperparameters of the support vector machines and their domains. For more information, see `https://scikit-learn.org/stable/modules/svm.html`.

The ONNX file format and extensibility

The design of the software was made in such a way that it is possible to extend the list of supported ML methods without breaking changes to the source code. This enables us to try different triggers in the future and compare their effectiveness. The key to the extensibility of the program is the uniform model format. After the training, each model is converted to an Open neural network exchange format. The ONNX format provides a uniform interface to heterogeneous models. Additionally, it serves as a bridge between the model's training (typically performed in Python programming language) and the performance-critical inference of the model (done in C++ language). In principle, it is possible to bypass the window trigger creator program and use an arbitrary ML model with custom training. However, it is very important to preserve the same input and output signature of the model and convert it to the ONNX file format. Nevertheless, it is best to consult such customization with the author.

Contact

Should you have ideas for improvement, problems, or questions related to the software, don't hesitate to get in touch with us at celko.tom@gmail.com. We hope that the developed tools will help you in your work.