GUI Development in openDIEL

Students: Yan Lam, Neptune Wong (City University of Hong Kong), Omar Tafiti (Morehouse College), Rocco Febbo (UTK)
Mentor: K. Wong (UTK)

What is OpenDIEL

OpenDIEL stands for Open Interoperable Distributive Executive Library. It is a wrapper to schedule work on a set of resources (workflow engine). It allows users to schedule tasks in parallel with helpful communication functionality. It also allows users to schedule tasks with dependencies.

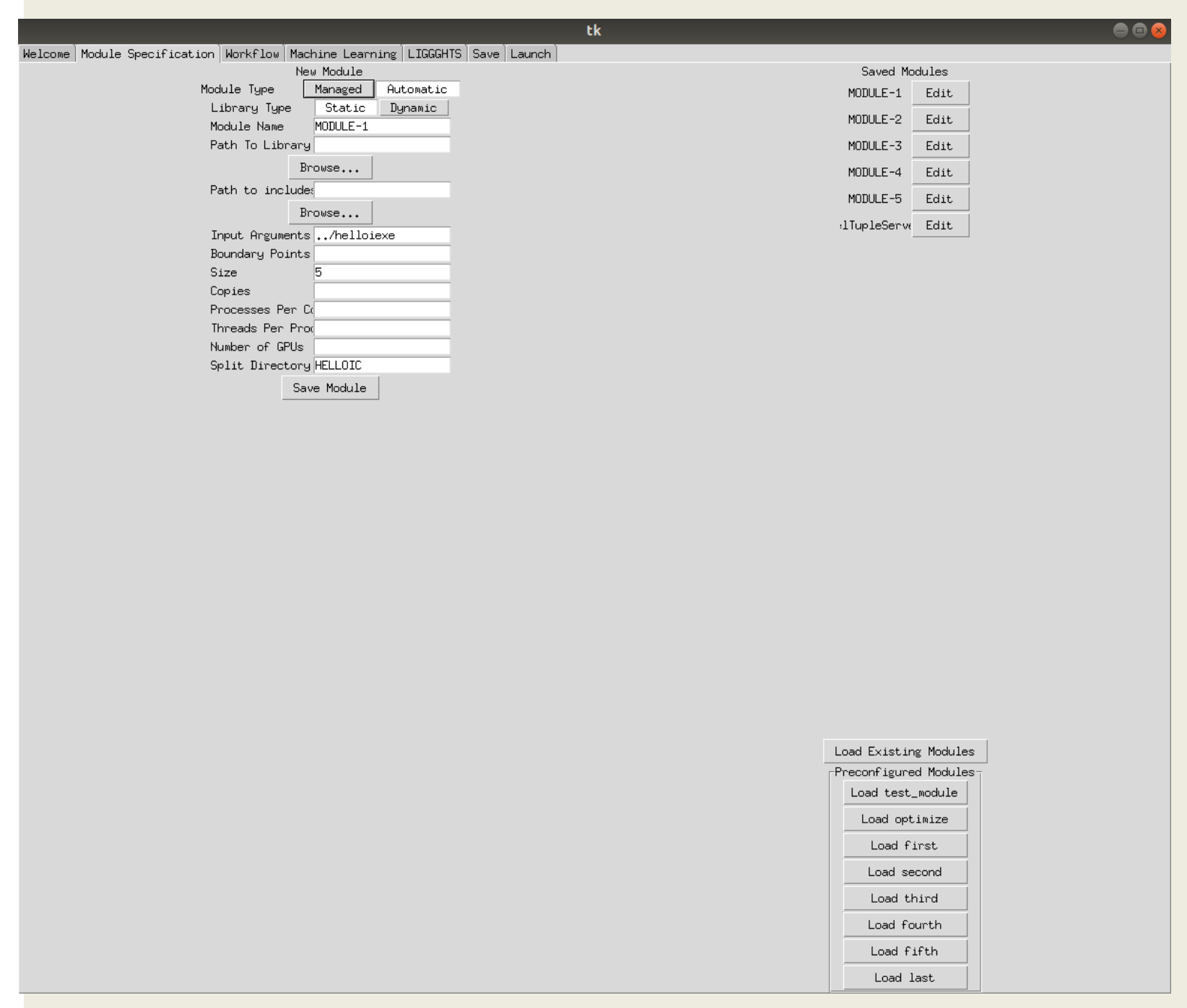
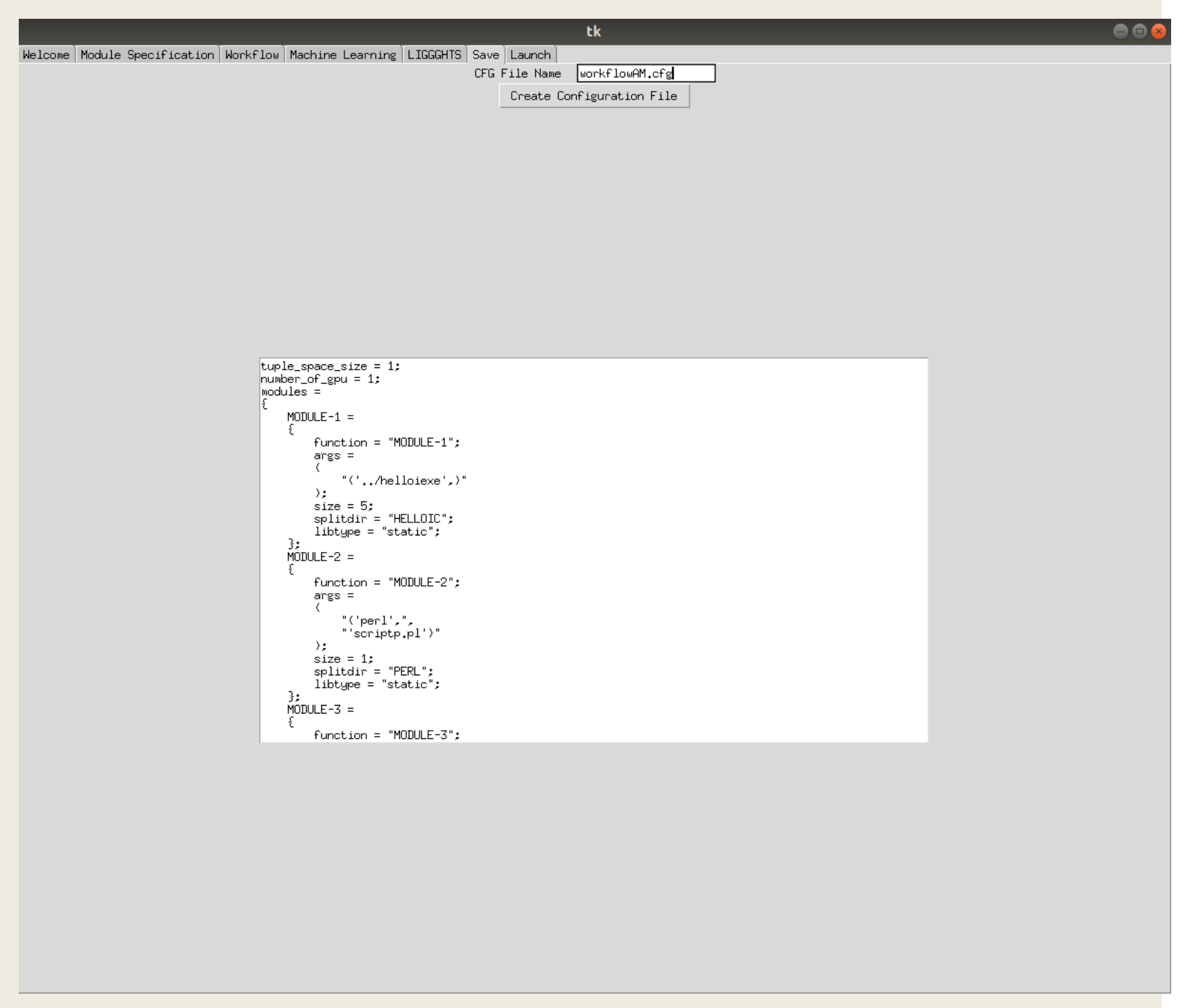
Problem with openDIEL

OpenDIEL is extremely resourceful but it can be tedious for users to operate.

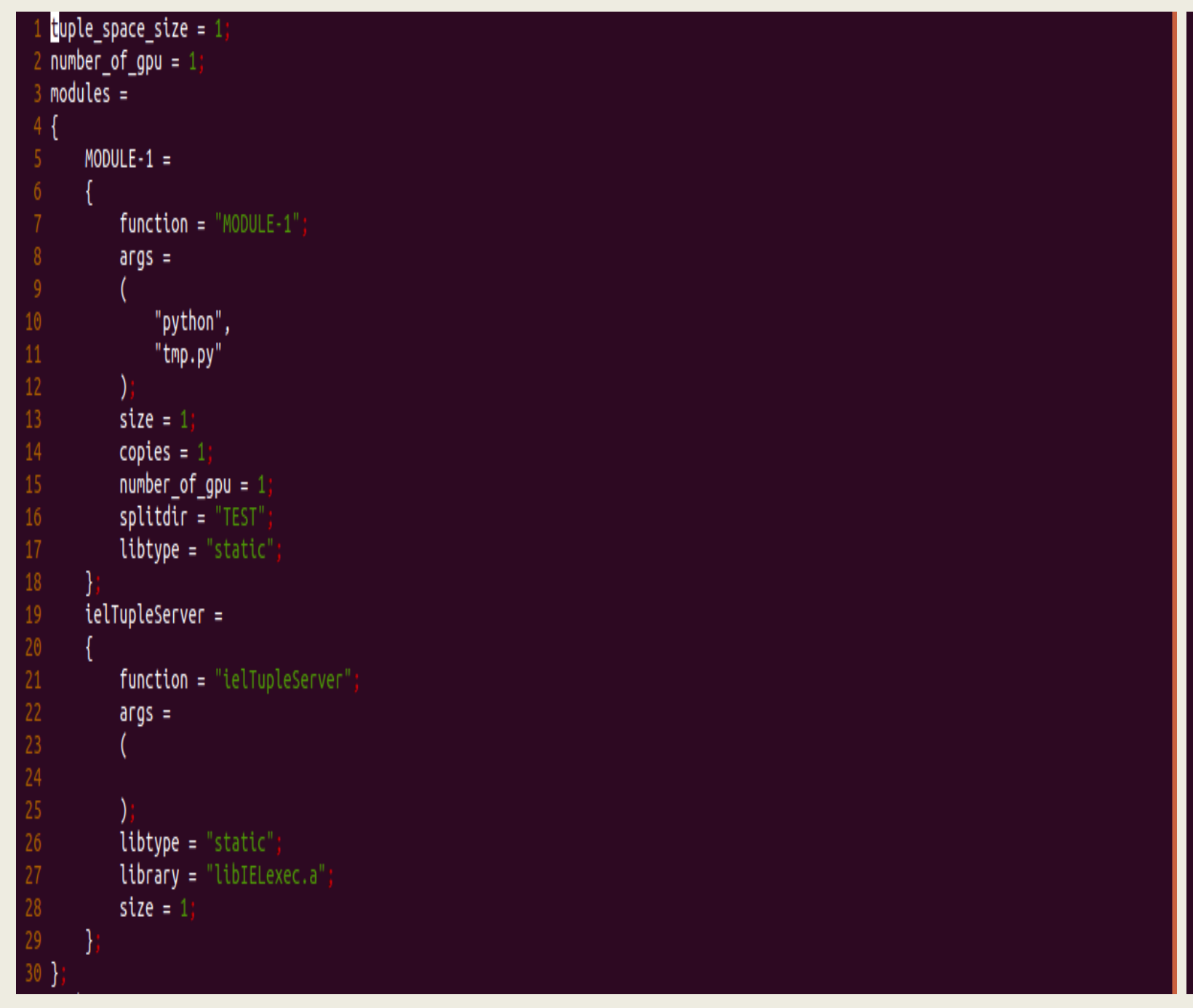
Research Goal

Because OpenDIEL can be tedious to operate we sought to create a fully functional Graphic User Interface(GUI) to help run tasks using OpenDIEL much simpler.

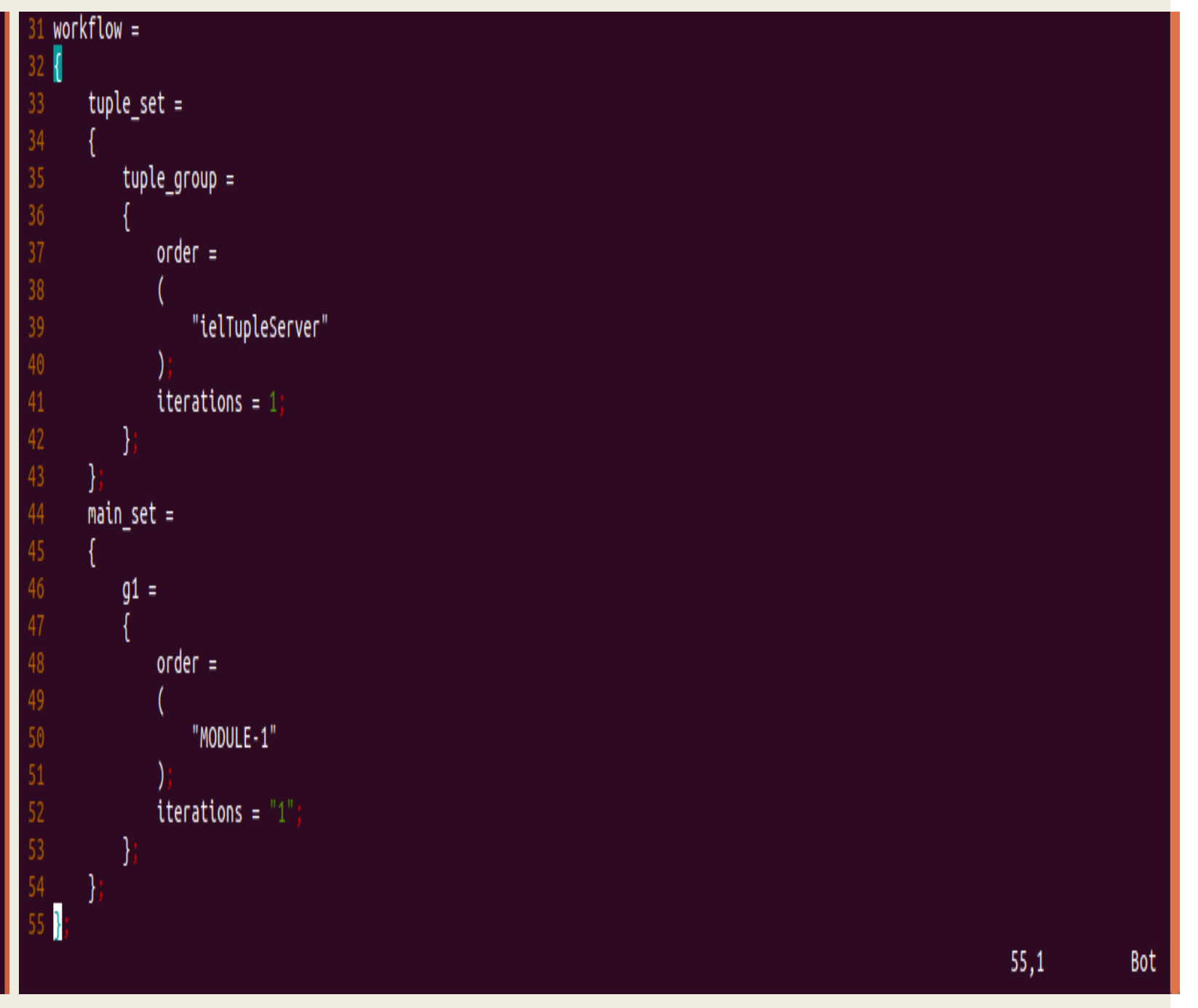
Tkinter

To create the GUI we have used tkinter which is a GUI Programming toolkit for Python.. The above picture is an example of creating widgets in Tkinter. This specific picture displays the widgets users can use to easily create modules or load existing modules to be ran with openDIEL. Once the user has either created or loaded their modules, they can proceed to create the workflow section for the modules. Then with the click of a button the configuration file that openDIEL uses will be created. The number of mpi processess will be calculated behind the scenes and the users example is ready to be launched.



```
tuple_space_size = 1;
number_of_gpu = 1;
modules =
{
    MODULE-1 =
    {
        function = "MODULE-1";
        args =
        (
            "python",
            "tmp.py"
        );
        size = 1;
        copies = 1;
        number_of_gpu = 1;
        splitdir = "TEST";
        libtype = "static";
    };
    ielTupleServer =
    {
        function = "ielTupleServer";
        args =
        (

        );
        libtype = "static";
        library = "libIELexec.a";
        size = 1;
    };
};
```



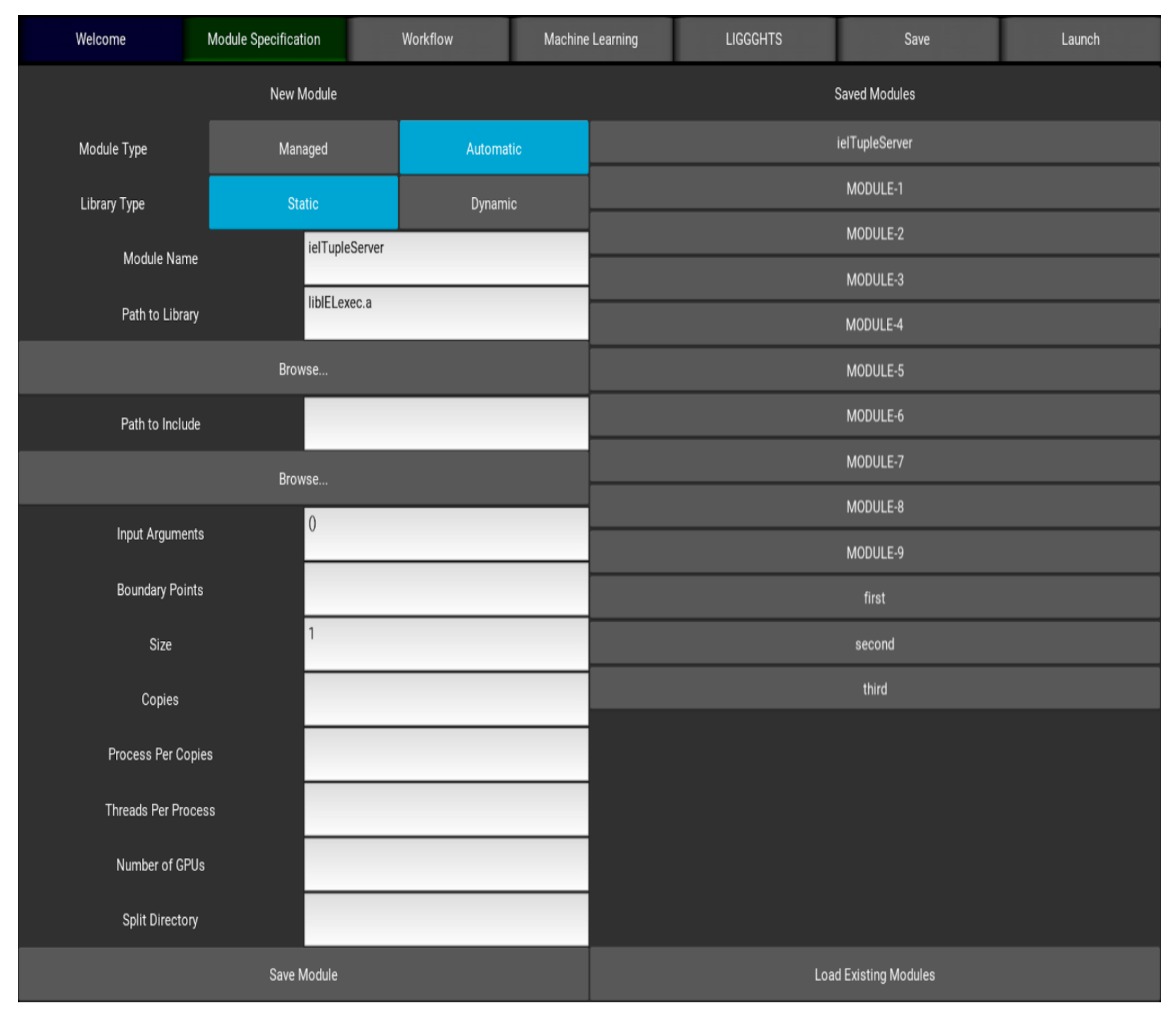
```
workflow =
{
    tuple_set =
    {
        tuple_group =
        {
            order =
            (
                "ielTupleServer"
            );
            iterations = 1;
        };
    };
    main_set =
    {
        g1 =
        {
            order =
            (
                "MODULE-1"
            );
            iterations = "1";
        };
    };
};
                                                55,1          Bot
```

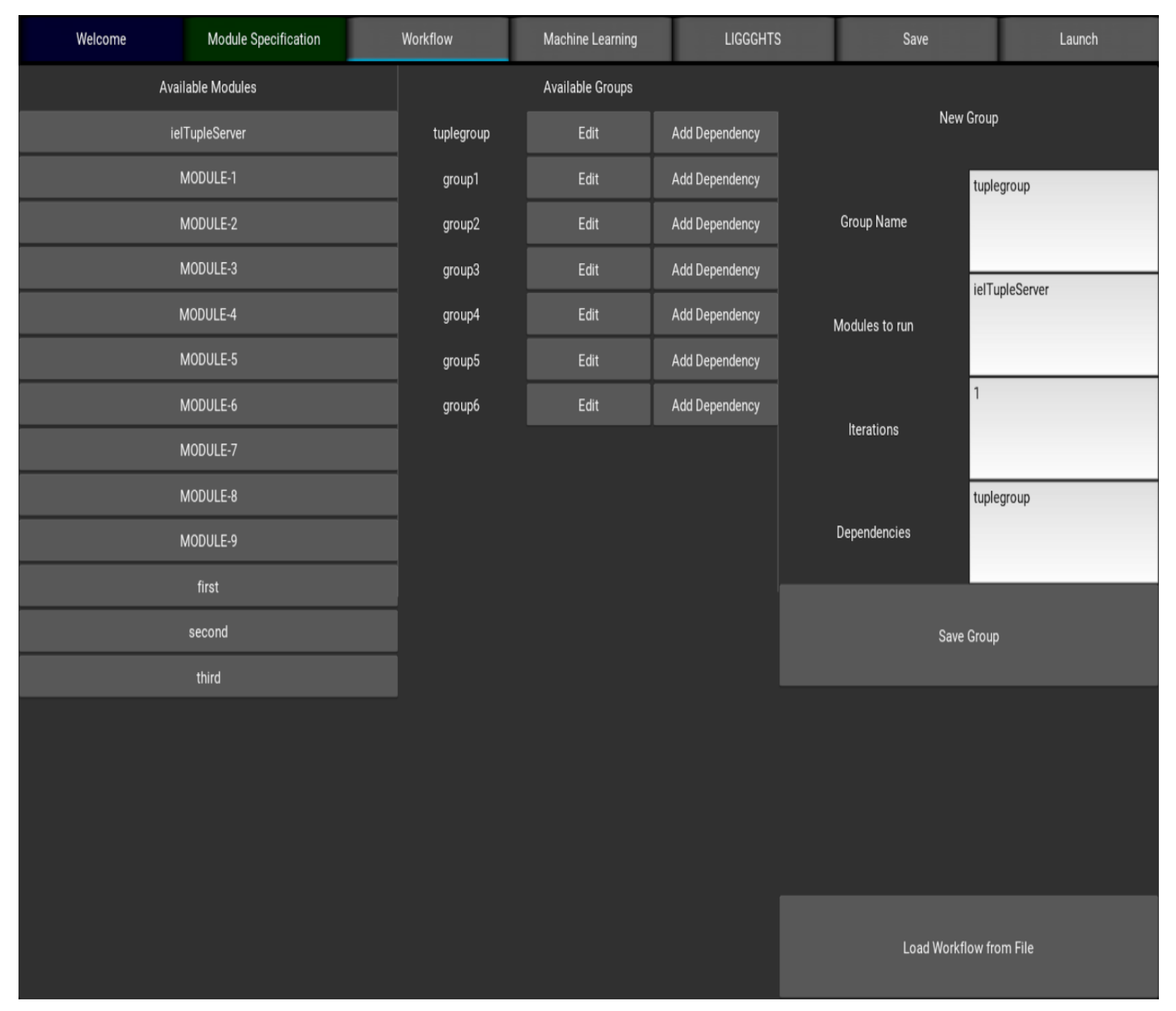
Problems with Tkinter

Although Tkinter is easy to use and highly efficient for creating GUI's, Tkinter is not the best when it comes to design and layout. Tkinter is also not as visually appealing as other GUI toolkits. It is desktop focused and can look outdated. Tkinter also does not have as many built in widgets as other GUI programming toolkits. We overcame some of these problems by using another GUI programming toolkit called Kivy.

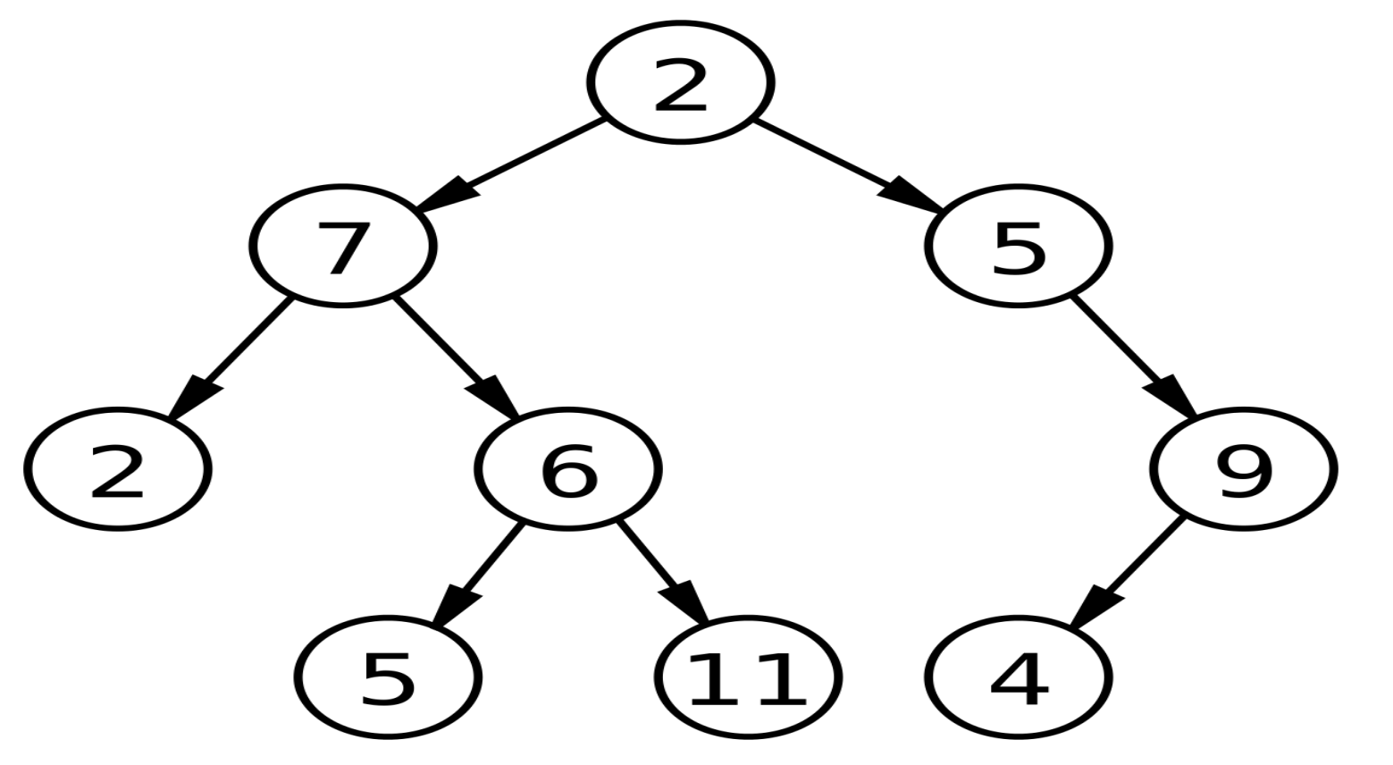
KIVY

Kivy is an open source Python framework which can make a innovative user interface for some applications that need rapid development.
Kivy is cross platform and GPU accelerated. it uses OpenGL ES 2 as the graphics engine which means it has a modern and fast graphics pipeline.

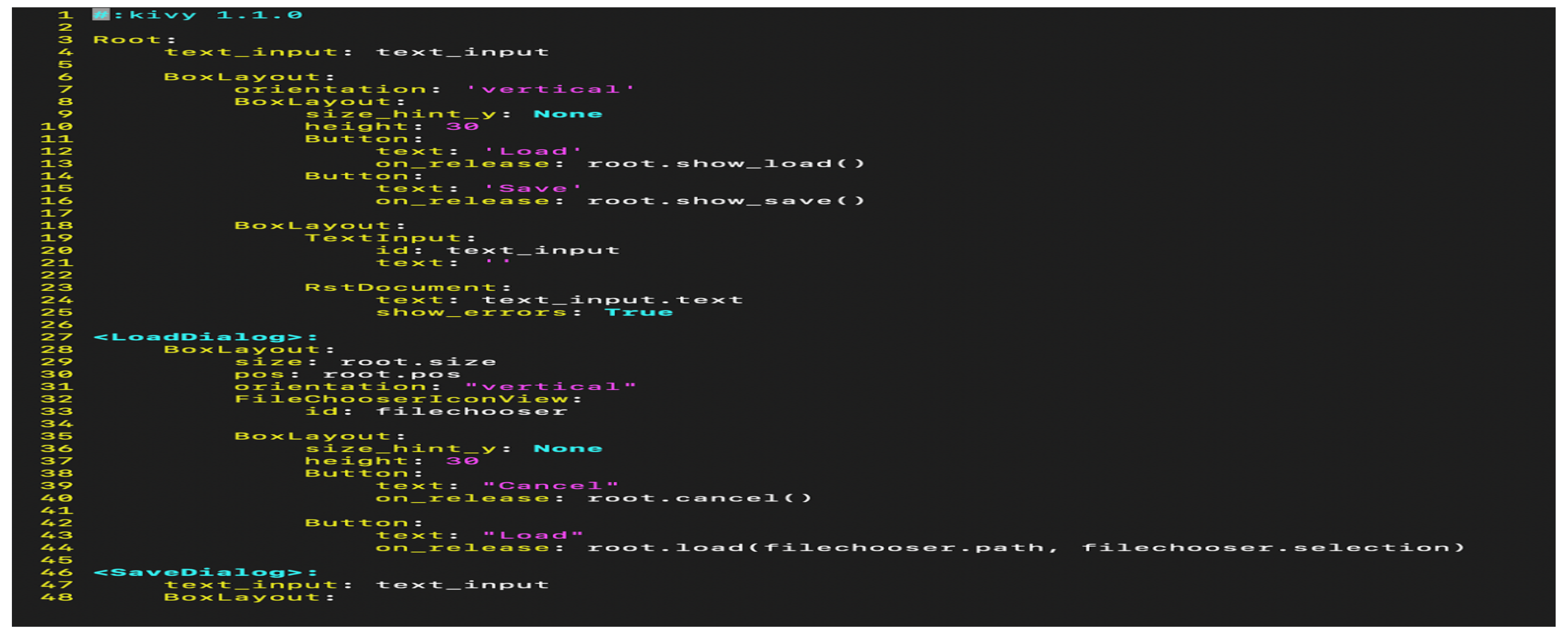




We have several tabs inside the application. The module tab is for inserting the module from a .cfg. Workflow is used for adding the module in to groups. Save module create the .cfg. whie Launch tab display the results after runing of the code.
Kivy works like a tree. It allows you to create your widget tree in a declarative way.
It also binds widget properties to each other or to callbacks in a natural manner



KIVY can automatically formats widgets that suitable for all platform. However, Kivy needs a special language to define the layout which allows logic to be keeped and seperate the presentation.

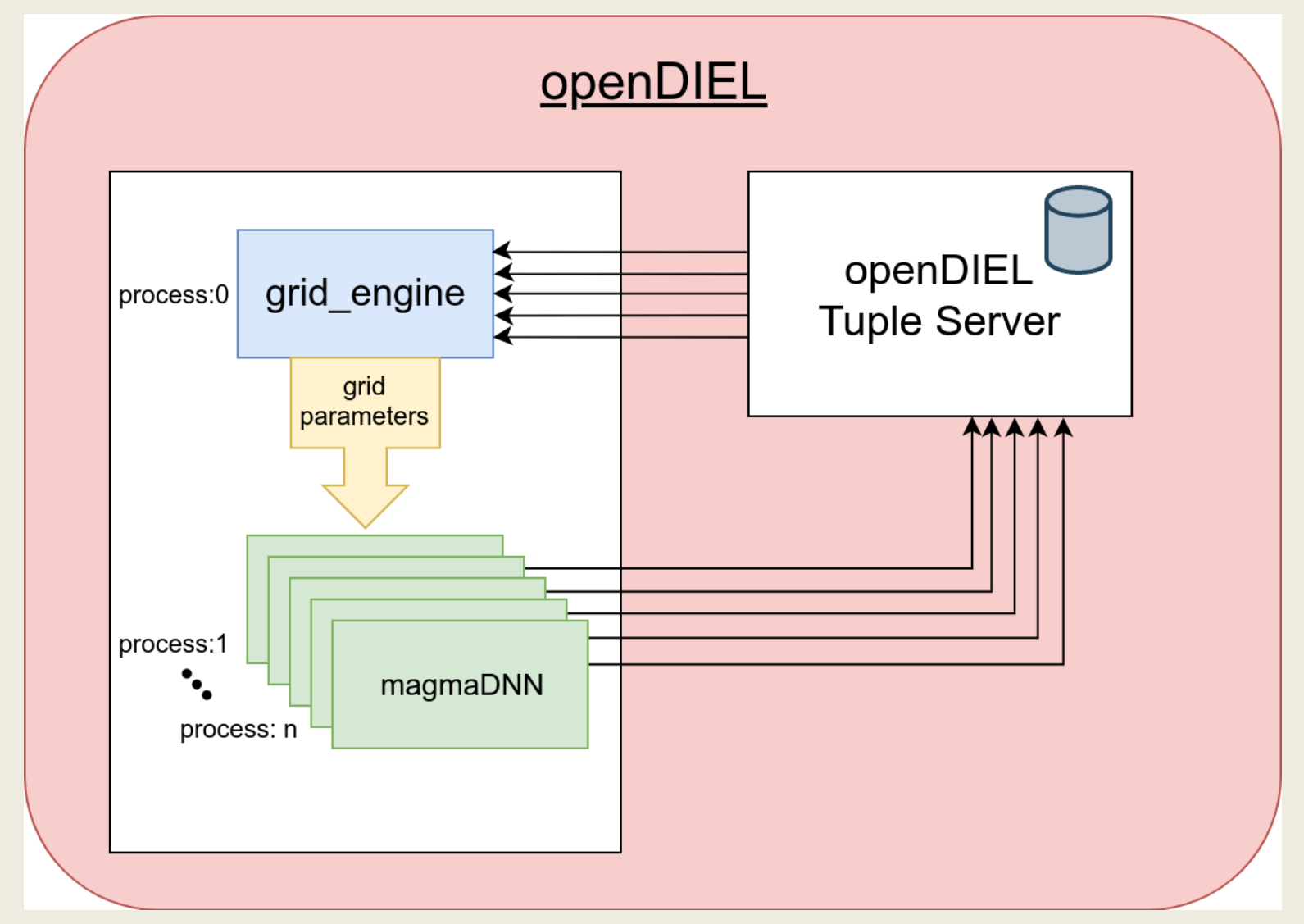


```
#:kivy 1.1.0

Root:
    text_input: text_input

    BoxLayout:
        orientation: 'vertical'
        BoxLayout:
            size_hint_y: None
            height: 30
            Button:
                text: 'Load'
                on_release: root.show_load()
            Button:
                text: 'Save'
                on_release: root.show_save()

        BoxLayout:
            TextInput:
                id: text_input
                text: ''

            RstDocument:
                text: text_input.text
                show_errors: True

<LoadDialog>:
    BoxLayout:
        size: root.size
        pos: root.pos
        orientation: "vertical"
        FileChooserIconView:
            id: filechooser

        BoxLayout:
            size_hint_y: None
            height: 30
            Button:
                text: "Cancel"
                on_release: root.cancel()

            Button:
                text: "Load"
                on_release: root.load(filechooser.path, filechooser.selection)

<SaveDialog>:
    text_input: text_input
    BoxLayout:
```

kv language is a language used to give the syntax of the kivy program a better view by representing all the elements in the program like classes, the other classes it is inheriting, widgets and their properties and configurations.
It is much more clear and understandable which means using kv language makes your code much more clear and organized. However, it is not always used (if the program is of very few lines). But it is a good approach to understand how things work in kivy.

Applications of OpenDIEL

Below is an illustration of a workflow created using OpenDIEL. It uses OpenDIEL's built in communication server, the Tuple Server, to pass information between different processes.

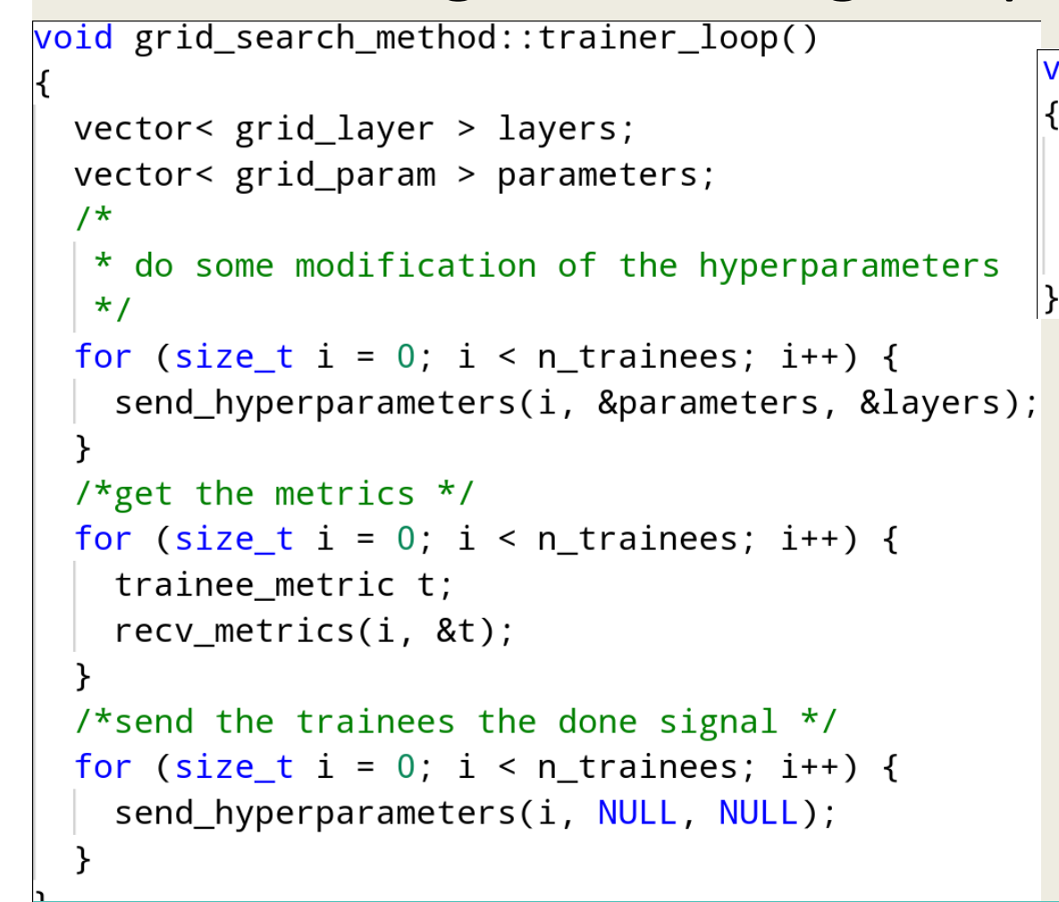


How Does the Tuple Server Work?

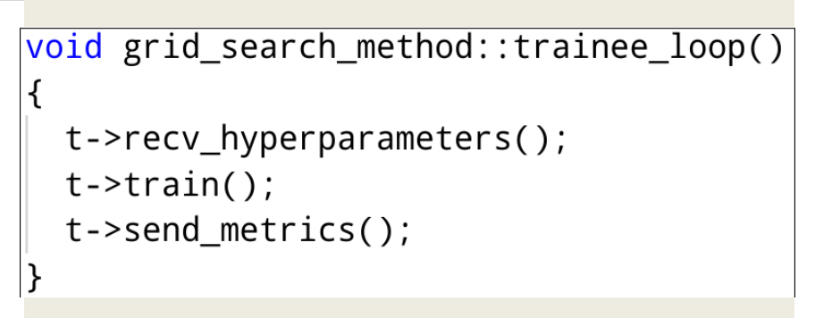
The Tuple Server is contained in it's own process. It acts like a storage container. Every piece of data is added to the Tuple Server along with a unique `tag` represented by an integer. The `tag` is how that data is then later accessed by other processes. _____

How Does the Grid Engine Work?

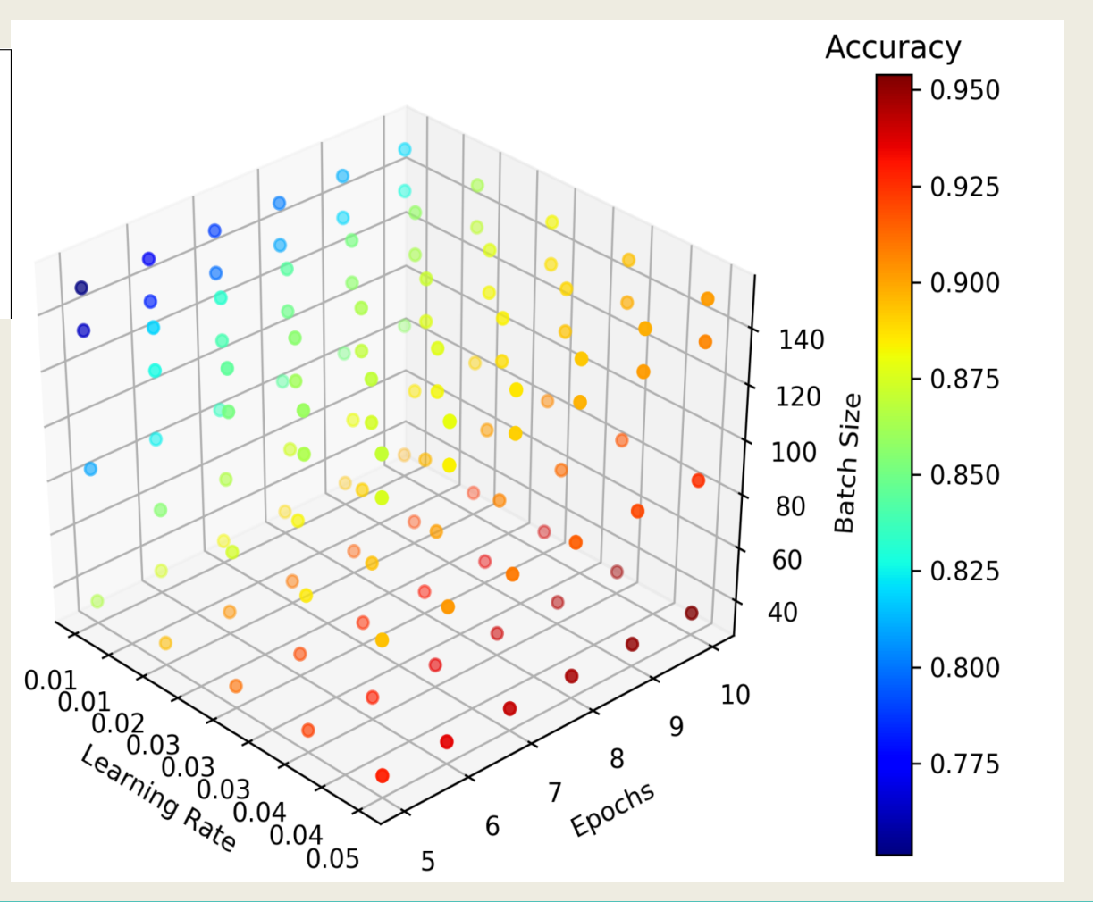
The Grid Engine manages a trainer and a set of trainees, each one its own process. The number of trainees depends on how big the OpenDIEL module size is and how many MPI process are allocated. The trainee sends hyperparameters to an OpenDIEL Tuple Server and the trainees receive that data, train, then report their accuracies to the Tuple Server. The trainer receives the accuracies and saves them to a file. It is designed to work with different search methods and different trainees. It has currently been tested using a grid search method and a MagmaDNN trainee. Below is an example of how the trainer can communicate with the trainees and the result after training over a 3D grid space.



```
void grid_search_method::trainer_loop()
{
  vector< grid_layer > layers;
  vector< grid_param > parameters;
  /*
   * do some modification of the hyperparameters
   */
  for (size_t i = 0; i < n_trainees; i++) {
    send_hyperparameters(i, &parameters, &layers);
  }
  /*get the metrics */
  for (size_t i = 0; i < n_trainees; i++) {
    trainee_metric t;
    recv_metrics(i, &t);
  }
  /*send the trainees the done signal */
  for (size_t i = 0; i < n_trainees; i++) {
    send_hyperparameters(i, NULL, NULL);
  }
}
```



```
void grid_search_method::trainee_loop()
{
  t->recv_hyperparameters();
  t->train();
  t->send_metrics();
}
```



Future Work

- Launch Tab of the kivy will be finished within week 9.
- Some of the revolution problems and the size of the layout need to have futher improvement.
- Color of the widget may need to changed.
- Add functionality to create custom parameters in the grid engine
- Add different types of search methods and trainees to the grid engine

Acknowledgements

This project is made possible only with the support by our mentor Dr. Wong, the NSF, the University of Tennessee and Oak Ridge National Laboratory.