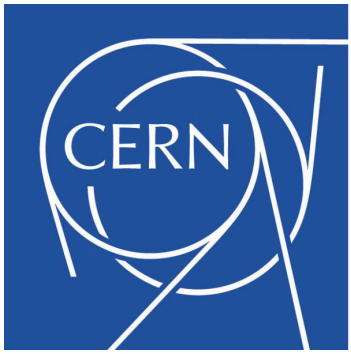


TMVA SOFIE: Enhancing Keras Parser and JAX/FLAX Integration

GSoC 2025 Project Proposal

Organization: CERN HSF

Mentors: Dr. Lorenzo Moneta lorenzo.moneta@cern.ch

Sanjiban Sengupta sanjiban.sengupta@cern.ch

Contact information:

Full name: Prasanna Sanjay Kasar

Degree: Bachelor of Technology in Computer Engineering (B. Tech)

University: Veermata Jijabai Technological Institute (VJTI), Mumbai

Location: Mumbai, Maharashtra, India

Time Zone: Indian Standard Time (UTC+5.30)

Contact: **Email** : prasanna.kasar06@gmail.com

GitHub : PrasannaKasar

LinkedIn : Prasanna Kasar

Resume: Link

Introduction

Hi, I'm Prasanna Kasar, and I'm currently in my second year of BTech in Computer Engineering at Veermata Jijabai Technological Institute, Mumbai.

I've developed several deep learning projects using Python, focusing on technologies like Transformers, CNNs, and RNNs, with frameworks such as PyTorch and TensorFlow. Here are a couple of my recent projects:

1. **Text-to-Speech:** Created a **Transformer-based Text-to-Speech** system, where I built a modular pipeline to convert raw text into **natural-sounding speech**. I used the Transformer's **Attention mechanism** in the front end to convert text to **mel spectrograms** and implemented the **Griffin-Lim algorithm** in the back end to reconstruct speech from the predicted **mel spectrograms**.

2. **Image Captioning:** Developed an **Image Captioning model** using **Inception v3** and **LSTM** on the **MS COCO dataset** to generate contextually relevant captions from images. Utilized **CNN** to extract **visual features** in the image and **LSTM** to process these features to generate coherent and descriptive captions.

I have a strong understanding of programming languages like Python, C/C++, and Java, and I'm also familiar with JavaScript. I've worked with C to solve various tasks and challenges given to me for this project.

I have worked with Tensorflow and have experience with its Sequential and Functional API while building several machine learning models.

Past Open Source Experience

I began my open-source journey with **Project X** at university, which closely resembles the Google Summer of Code (GSoC) process. I worked on a Text-to-Speech system, gaining experience with libraries like **PyTorch**, **Librosa,** and **NumPy,** and learned how to use **Git** and **GitHub** effectively.

After completing these projects, I participated in **HacktoberFest 2024**, merging all four pull requests before the deadline.

During this year's GSoC, I encountered challenges while building **ROOT** on Windows, so I switched to **Linux**. This made the process easier and provided me with greater control over the system.

Time Commitment

I can dedicate 7 to 8 hours per day to the project, which would amount to 49-52 hours per week. I am open to increasing this duration if needed. As I do not have any strict commitments during this summer, I can easily adjust my schedule and work for longer hours if required. I am flexible with working hours and can adapt them according to my mentor's time zone. My summer vacation is from May 31 to July 31, so a significant portion of the work can be completed during this period.

Pre-GSoC Evaluation Tasks

I have submitted the **preliminary tasks**, as well as the **project-specific tasks,** to the Project Mentors Dr. Lorenzo Moneta and Sanjiban Sengupta.

GitHub **personal development branch** of ROOT: Link

Preliminary task document: Link

Enhancing Keras Parser and JAX/FLAX Integration task document: Link

Project Description

The **SOFIE (System for Optimized Fast Inference Code Emit)** project is an initiative within the **TMVA (Toolkit for Multivariate Data Analysis)** framework in ROOT, which aims to enhance the efficiency and speed of inference on Machine Learning Models. SOFIE converts ML models trained in different frameworks such as ONNX, PyTorch, and Tensorflow, converting them into an **Intermediate Representation (IR)**. This IR allows SOFIE to generate optimized C++ functions for fast and effective inference of neural networks and subsequently convert them into C++ header files, which can be used in plug-and-go style for inference.

For GSoC 2025, this project aims to **enhance the Keras parser** to support models trained in the latest TensorFlow v2.18.0, which introduces NumPy 2.0 compatibility. As well as **Integrate JAX/FLAX support**, enabling SOFIE to generate C++ inference functions for models developed using JAX/FLAX

Project Milestones

Researching the latest TensorFlow/Keras: Investigate about the latest TensorFlow/Keras developments (version >= 2.18) since it supports NumPy 2.0 now.

Improving the Keras Parser: Implement Keras parser with support for latest Tensorflow/Keras and NumPy versions.

JAX/FLAX Integration: Investigating alignment JAX/FLAX libraries with SOFIE and developing parser to parse models trained with the same.

Current Keras Parser in SOFIE

There's already a **Keras** parser in **SOFIE** which is independent of the **ONNX** Parser. The reason for it is:

1. There is no simple converter between **Keras** and **ONNX**, unlike **Pytorch** where a model trained in Pytorch is first converted to ONNX format and then it is parsed via ONNX Parser.
2. Keras supports some functions that ONNX does not (for example the **Swish** activation function)

Because of these reasons, the Keras parser is developed separately. However, the current Parser has few issues:

1. It only supports **Dense** and **Convolutional** layers (along with their activation functions), but not **Pooling** and **Recurrent** layers (such as RNN, LSTM, GRU)
2. It is written in **C++**, while most of the parsing is actually done in **Python**.

```
'history',
'input',
'input_dtype',
'input_shape',
'input_spec',
'inputs',
'jit_compile',
```

Mode attributes of Keras model

```
// will have their dtype as string
PyRunString("inputNames=model.input_names",fGlob
PyRunString("inputShapes=model.input_shape if ty
PyRunString("inputTypes=[]",fGlobalNS,fLocalNS);
```

(Missing input_names attribute for model object)

```
RModel Parse(std::string filename, int batch_size){

   // Extracting model information
   // For each layer: type,name,activation,dtype,input tensor's name,
   // output tensor's name, kernel's name, bias's name
   // None object is returned for if property doesn't belong to layer
   PyRunString("import tensorflow",fGlobalNS,fLocalNS);
   PyRunString("import tensorflow.keras as keras",fGlobalNS,fLocalNS);
   PyRunString("from tensorflow.keras.models import load_model",fGlobalNS,fLocalNS);
   PyRunString("print('TF/Keras Version: '+ tensorflow.__version__)",fGlobalNS,fLocalNS);
   PyRunString(TString::Format("model=load_model('%s')",filename.c_str()),fGlobalNS,fLocalNS);
   PyRunString(TString::Format("model.load_weights('%s')",filename.c_str()),fGlobalNS,fLocalNS);
   PyRunString("globals().update(locals())",fGlobalNS,fLocalNS);
   PyRunString("modelData=[]",fGlobalNS,fLocalNS);
   PyRunString("for idx in range(len(model.layers)):\n"
               "  layer=model.get_layer(index=idx)\n"
               "  layerData={}\n"
               "  layerData['layerType']=layer.__class__.__name__\n"
               "  layerData['layerAttributes']=layer.__dict__\n"
               "  layerData['layerInput']=[x.name for x in layer.input] if isinstance(layer.input,list)
               "  layerData['layerOutput']=[x.name for x in layer.output] if isinstance(layer.output,li
               "  layerData['layerDType']=layer.dtype\n"
               "  layerData['layerWeight']=[x.name for x in layer.weights]\n"
               "  modelData.append(layerData)",fGlobalNS,fLocalNS);
```

(Parsing models trained with Tensorflow using Python in C++ Keras Parser of SOFIE)

3. It does not support models trained with latest **Tensorflow** and **NumPy** versions

```
Python error message:
Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "/home/prasanna/RootDevelopment/py310/lib/python3.10/site-packages/keras/
src/saving/saving_api.py", line 238, in load_model
    return legacy_sm_saving_lib.load_model(
  File "/home/prasanna/RootDevelopment/py310/lib/python3.10/site-packages/keras/
src/utils/traceback_utils.py", line 70, in error_handler
    raise e.with_traceback(filtered_tb) from None
  File "/home/prasanna/RootDevelopment/py310/lib/python3.10/site-packages/keras/
src/engine/base_layer.py", line 870, in from_config
    raise TypeError(
TypeError: Error when deserializing class 'InputLayer' using config={'batch_shap
e': [None, 28, 28], 'dtype': 'float32', 'sparse': False, 'ragged': False, 'name'
: 'input_layer'}.

Exception encountered: Unrecognized keyword arguments: ['batch_shape']
Error in <TRint::HandleTermInput()>: std::runtime_error caught:
Failed to run python code: model=load_model('/home/prasanna/RootDevelopment/root
/tutorials/machine_learning/new_keras_model.h5')
root [5]
```

(Error while generating inference for Keras model trained with Tensorflow 2.18 and NumPy 2.1.3)

New Keras Parser in Python

Because of these reasons, in this project I aim to develop a new **Keras** Parser, which would support models trained with newer Tensorflow and NumPy versions, to be written entirely in **Python**.

How would a Python Parser work with SOFIE if everything from Parsing the model till making the inference is associated with C++?

To use SOFIE's Intermediate Representation (RModel) with Python and to make use of already implemented ROperators for ONNX, we can use **PyROOT** and **Pythonization.**

PyROOT

PyROOT is a Python interface for the ROOT framework, which enables seamless interaction between Python and ROOT's C++ libraries by dynamically creating bindings between the two languages. This allows us to access the full functionality of ROOT's C++ classes (including SOFIE) and functions directly from Python.

```
import ROOT
import numpy as np


ROOT.TMVA.PyMethodBase.PyInitialize()


# check if the input file exists
modelFile = "Higgs_trained_model.h5"
if (ROOT.gSystem.AccessPathName(modelFile)) :
    ROOT.Info("TMVA_SOFIE_RDataFrame","You need to run TMVA_Higgs_Classifica
    exit()


# parse the input Keras model into RModel object
model = ROOT.TMVA.Experimental.SOFIE.PyKeras.Parse(modelFile)

generatedHeaderFile = modelFile.replace(".h5",".hxx")
print("Generating inference code for the Keras model from ",modelFile,"in th
#Generating inference code
model.Generate()
model.OutputGenerated(generatedHeaderFile)
```

(Example: Use PyROOT for generating header file for Higgs mode using current C++ parser for Kerasl)

Pythonization

Pythonization is a Decorator that allows Pythonize C++ classes. Pythonize means to add some extra behaviour to a C++ class that is used from Python via PyROOT, so that such a class can be used in an easier / more pythonic way. This helps ROOT's C++ classes and functions to align better with Python's syntax and conventions, improving usability.

Steps while Developing the Keras Parser

1. **Create RModel object**

Initialize a RModel object using `TMVA.Experimental.SOFIE.RModel.RModel` to store the Intermediate Representation while parsing and return it for further inference.

2. **Extract relevant features from the model**

A ML model consists of various layers. And each of these layers has these 2 properties:

1) **Operator** (RNN, LSTM, Softmax, ReLU, Tanh, etc)
2) **Layer weights** or **Trainable parameters**

Few Layers are made up of **Operators** + Their **Activation functions**, such as **Dense**, **Convolution**, etc.

We need to extract these features from the model and store them in the form of a **Python Dictionary.** To do this, we Iterate over each layer of the model, extract features such as **layerType**, **layerAttributes**, **layerInput**, **layerOutput**, **layerDType**, **layerWeight**. Few operators, such as **RNN**, have more attributes, like: activation, hidden size. We need to add an `if` clause for these layers while extracting the features.

3. **Create functions to parse the layers**

We create functions for each of the layers to parse them. These functions would return the operator object from class `ROperator_[op_name]`, which are already implemented in C++. We would do this by leveraging **PyROOT** and **Pythonization**.

```python
def ParseKerasTanh(layer):

  InputName = layer['input_name']
  OutputName = layer['output_name']

  tanh_op = TMVA.Experimental.SOFIE.ROperator_Tanh(InputName, OutputName)

  return tanh_op
```

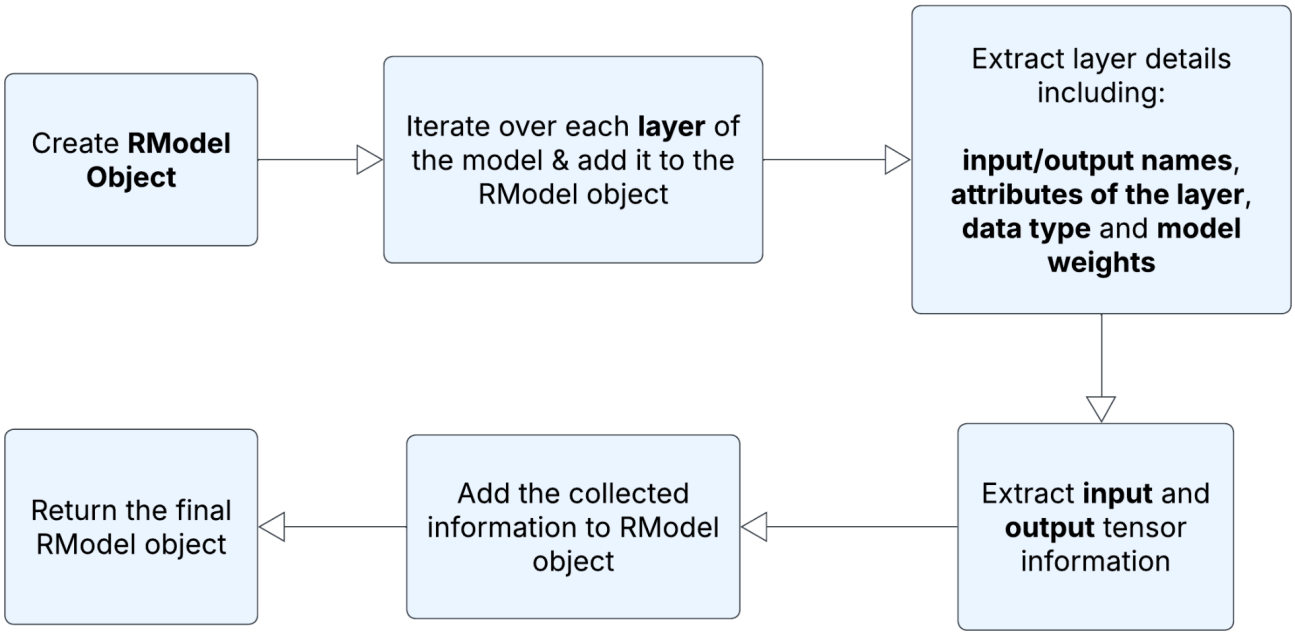
(Example: Parsing function for Tanh operator (abstract))

4. **Save layer weights & Input-Output tensor information**

Similar to extracting layer attributes, we extract the l**ayer weights** by iterating over each layer of the model. Along with it, we save the **Input** and **Output** tensor dimension of the model.

For each layer, after extracting the relevant information, add it to the **RModel object** and return it. This returned RModel object can be used to generate inference code by using the `Generate` function inside SOFIE.

Parser Workflow



SOFIE Parser for JAX/FLAX

FLAX is a neural network library which is built on top of **JAX**. JAX is similar to **NumPy** but provides additional features such as **immutable arrays** and **automatic differentiator** for Gradient descent and Backpropagation and it is faster than NumPy.

Similar to Keras/Tensorflow, there is no simple converter between **FLAX** and **ONNX**. So, we need to develop a **Python** parser for FLAX as well.

Model classes used in FLAX

There are 2 different model building APIs used with FLAX to build the Neural Networks:

1. **Flax.linen** (Most commonly used)
2. **Flax.nnx** (Relatively newer)

Both of the classes have different attributes.

```
['count',
 'eval',
 'iter_children',
 'iter_modules',
 'linear1',
 'linear2',
 'perturb',
 'set_attributes',
 'sow',
 'train']
```

Flax.nnx

```
['apply',
 'bind',
 'clone',
 'copy',
 'get_variable',
 'has_rng',
 'has_variable',
 'init',
 'init_with_output',
 'is_initializing',
 'is_mutable_collection',
 'lazy_init',
 'make_rng',
 'module_paths',
 'name',
 'param',
 'parent',
 'path',
 'perturb',
 'put_variable',
 'scope',
 'setup',
 'sow',
 'tabulate',
 'unbind',
 'variable',
 'variables']
```

Flax.linen

(Model attributes of Flax.nnx and Flax.linen)

The parser designed for parsing FLAX models would be similar to Keras Parser which would be written in **Python**. It would extract relevant features from model layers such as its size, shape, type, layer input, layer output, layer weights, etc. But the only difference would be that it would extract essential information for models built with **linen** and **nnx** API separately.

In the first part, I aim to develop the parser for models trained in JAX/FLAX's **Linen** API, and then with its **nnx** API.

Existing Python Parser for Keras

There already exists a Keras parser written in python by **Uri Stern** in **August 2023**. It supports nearly every operator of Keras/Tensorflow.

However, since it was developed in **2023**, it does not support **NumPy 2.0**, since NumPy 2.0 was released in **November 2024**.

I would like to extend the support of this parser for **NumPy 2.0**. Additionally, I would like to add support for parsing operators such as the **Transformer** module and **2D Convolution transpose**, which are missing in the Python parser.

Project Timelines

Time Period	Task
Up until May 8: Proposal Acceptance or Rejection	➔ Become familiar with the codebase of SOFIE and ROOT. ➔ Analyze current Parser capabilities of Keras. ➔ Discuss any doubts related to implementation with your mentor. ➔ Establish a proper project structure to ensure smooth development. **Deliverable**: Gain solid knowledge of parsing capabilities of the current Parsing mechanism for Keras models and have a clear understanding of the project's requirements and objectives, with no remaining doubts.

May 8 - June 1: Community Bonding Period	➔ Get a clear understanding of the current C++ and Python parser for Keras. ➔ Get familiar with JAX/FLAX libraries and their various Neural Network APIs to build Deep Learning Models. **Deliverable**: Have a clear understanding of the current Keras parser and its functionalities.
Week 1: Official Coding Begins	➔ Start working with Keras Parser in Python. ➔ Research the latest developments with TensorFlow and NumPy 2.0. ➔ Discuss with mentors regarding the alignment of the latest TensorFlow and NumPy with SOFIE. **Deliverable**: Have a clear understanding of the changes required to support the latest version of TensorFlow with NumPy 2.0 compatibility.
Week 2-3	➔ Study the parsing mechanism to extract relevant features of the Keras model trained with TensorFlow 2.18. **Deliverable**: Create a Python function to extract essential features from the layers of the Keras model.
Week 4	➔ Explore PyROOT and Pythonization to use ONNX ROperators inside Python. **Deliverable**: Write functions to return the ROperator objects for respective operators of Keras.
Week 5	➔ Examine the present tests written for C++/Python parsers and add similar custom tests to validate the developed parser. **Deliverable**: Write custom tests for parsing Keras models with various types of layers and verify the generated C++ code.
Week 6	**Buffer period** ➔ Complete support for parsing remaining operators.

	➔ Prepare documentation for developed TensorFlow/Keras parser for mid-term evaluation.
Week 7	➔ Start working with Python Parser to parse models built with JAX/FLAX libraries. ➔ Explore FLAX's linen as well as nnx module. ➔ Discuss with mentors regarding its potential integration with SOFIE. **Deliverable**: Have a clear understanding of the parser required to parse JAX/FLAX models.
Week 8, 9, 10	➔ Study the parsing mechanism to extract relevant features of the FLAX models built using various Neural Network APIs of FLAX. ➔ Start Implementation of parsing mechanism for FLAX's linen API ➔ After completing the parser for linen API, start working with parser for FLAX's nnx API. **Deliverable**: Create a Python function to extract essential features from the layers of FLAX models that support all types of FLAX Module APIs.
Week 11	➔ Explore PyROOT and Pythonization to use ONNX ROperators inside Python. ➔ Examine the present tests written for C++/Python parsers and add similar custom tests to validate the developed parser. **Deliverable**: Write functions to return the ROperator objects for respective operators of Keras. Write custom tests for parsing Keras models with various types of layers and verify the generated C++ code.
Week 12	➔ Prepare documentation for developed FLAX parser. ➔ Ensure all mentor requirements are fulfilled. ➔ Focus on documentation and prepare a blog post. ➔ Summarize all work in a blog post and submit it for evaluation.

Why me and my Motivation?

I found SOFIE's method of converting trained ML models into C++ header files for fast inference to be particularly impressive. My goal is to add support for models trained with the latest versions of Tensorflow/Keras and FLAX.

I have always aspired to work for CERN and would be highly grateful to receive the opportunity to contribute as a Google Summer of Code student developer. This is a great opportunity and a head start for achieving success in my life.

References

Python Parser for Keras in SOFIE
Introduction to JAX/FLAX (Presentation)
FLAX:nnx Module Attributes