International Journal of Innovative Research in Computer Science & Technology (IJIRCST)
ISSN: 2347-5552, Volume-10, Issue-2, March 2022
https://doi.org/10.55524/ijircst.2022.10.2.14
Article ID IRPV1035, Pages 72-76
www.ijircst.org

# PyKaldi: A Kaldi Wrapper in Python

**Jasdeep Singh,**

Assistant Professor Department of Computer Science & Engineering, RIMT University, Mandi Gobindgarh, Punjab, India

Correspondence should be addressed to Jasdeep Singh; jasdeepsingh@rimt.ac.in

**ABSTRACT:** PyKaldi is much more than a set of Kaldi library bindings. It offers best level of compatibility for OpenFst classes with the tool "Kaldi" to make dealing with Kaldi easier for Python users. PyKaldi is wrapper most probably written in a language known as "Python" for the widely used Kaldi SR toolkit that is free and open-source. PyKaldi isn't only a set of Python bindings for Kaldi libraries. It's a Python-based coding that lets programmers or developers interact with OpenFst types or Kaldi in real time. NumPy arrays are strongly integrated with both of the tools discussed. PyKaldi, we hope, will substantially improve the user experience and make integrating Kaldi into Python processes much easier. PyKaldi has a lot of documentation and testing. It supports Python 3 and also the previous version 2, and is distributed under the Apache License version 2.0. The fact that Kaldi has been so efficacious should not arise as any wonder. The features related to licence, rich documentation, tried methods for developing cutting-edge systems, a big number of international contributors, a devoted set of maintainers, and, perhaps most significantly, a well-designed codebase that is simple to comprehend, alter, and expand.

**KEYWORDS** CPython, Cython, Kaldi, OpenFst, PyKaldi, Python, Speech Recognition.

## I. INTRODUCTION

Kaldi is a voice recognition toolkit that is free and open-source. It includes a huge number of sample scripts for creating systems, as well as contemporary, stretchy, wide-ranging archives and executable set of programs developed in C++ as shown in Fig. 1[1-3]. It has quickly become an important utility for doing spoken language experiments and developing spoken-language-assisted apps since its introduction in 2011. Users usually intermingle with Kaaldi by manually compiling & executing its extremely modular and composable terminal applications within any of the Linux or UNIX terminal or by developing programs that execute these programmes. The C++ API can be used to access any feature not offered as a result of the several command-line Kaldi applications. While this interaction approach is quite successful, it falls short of meeting the demands of academics or the programmers who want to utilise Kaaldi in programming languages apart from C++ [4-6].

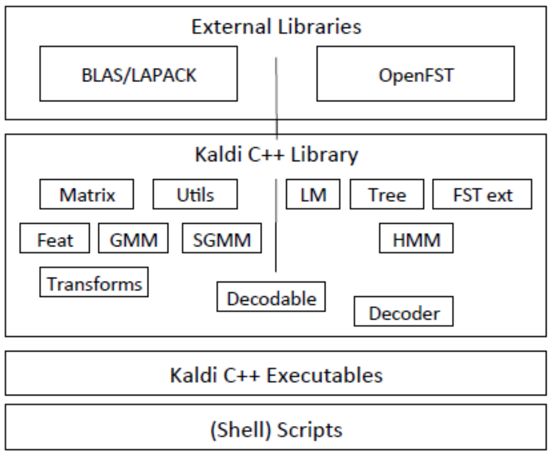


Figure. 1: Illustrates the architecture of Kaldi [1]

Python is a widely used general-purpose high-level programming language in the analytical computing field [7-8]. It features a straightforward syntax, a large standard library, and a well-developed community of incredibly great quality third-party tools for nearly any application, including numerical computation and deep learning [9-10]. One of the greatest platforms for interactive experimentation, data analysis, and visualisation is included. In addition, the benchmark CPython implementation offers a C language API for developing innovative in-built datatypes & integrating in the available libraries of C programming language as shown in Fig. 2, which may be used to offload results work to C/C++ with excellent success [11-14]. Python attachments for Kaaldi modules are one amongst most requested improvements among Kaldi users for all of these factors and more. There are several open-source programmes that attempt to narrow the gap among Kaldi and Python, however they are all quite restricted in scope.

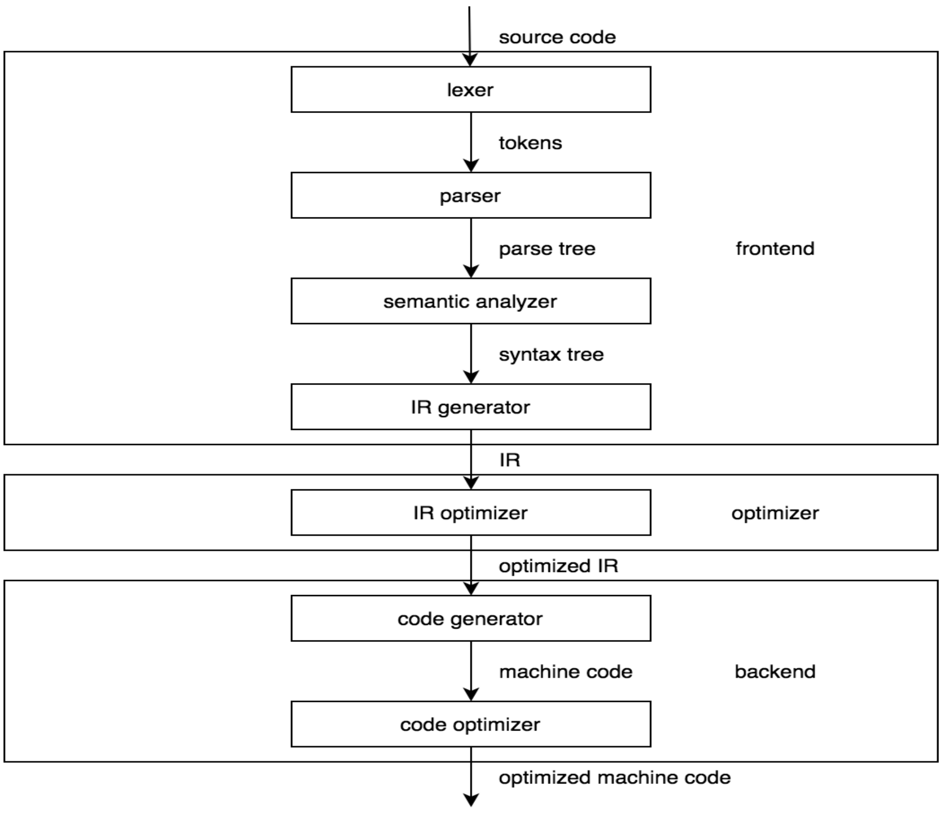


Figure. 2: A model of a vintage CPython compiler is depicted in the image above [11]

PyKaldi, an open and free scripting layer as depicted in Fig. 3 for Kaldi, is reviewed in this research study, and it offers a closer analysis of Kaaldi's C++ framework programming interface in Python [15-19]. Using graphical Python interpreters like IPython such kinds may be readily created, edited, and shown. By exchanging the fundamental storage buffers, PyKaldi utility types may be transformed to NumPi arrays and conversely [20]. The PyKaldi FST or transducers categories, which include style of Kaldi lattices, have an API that is comparable to that of OpenFst's original Python wrapper. PyKaldi helps make interacting with Kaldi in Python a snap, even if it is still in its early stages. The following are some of PyKaldi's most notable features:

- Kaldi is almost completely covered.
- The design is adaptable:
- PyKaldi is a flexible and easily-retainable programming language. The class pecking order of Kaldi and OpenFst are enfolded at several stages in Python, providing non-specific interfaces. Any modifications to the Kaaldi C++ interface may be simply replicated in other languages too.
- Unrestricted license:
- PyKaldi is released under the Apache 2.0 License.
- There is a lot of documentation:
- A variety of sub-modules in PyKaldi already have substantial documentation. The documentation for all APIs is produced dynamically from source code. Furthermore, as much of PyKaldi's API is a direct replica of Kaldi, almost all of PyKaldi's reference is relevant to Kaldi.
- Extensive testing:
- A lot of sub-modules in PyKaldi already have thorough testing. These tests are similar to Kaldi's, except they additionally include checks for more Python APIs.
- Script examples:
- The PyKaldi repository contains sample Python scripts that may be used to replace certain Kaldi executables. We're also focusing on example configurations that show how PyKaldi may be used in conjunction with famous Python modules.
- Python 3 and the earlier version 2 are both supported.

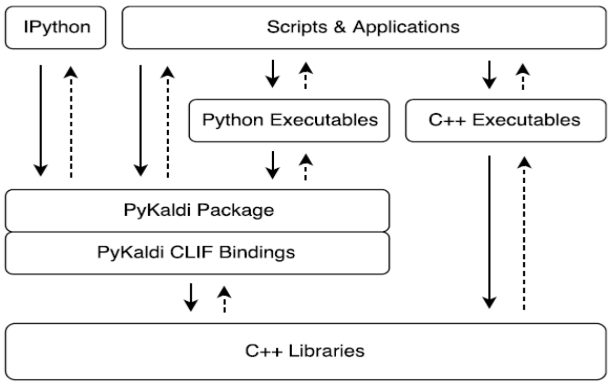


Fig. 3. The picture above shows extended Kaldi software architecture [21]

## II. DISCUSSION

### *A. CLIF bindings*

When it comes to developing Python bindings for C++, there are a lot of choices. Improved Wrapper and Interface Builder (IWIB) is an earlier and more developed project. It links C and C++ applications to a variety of languages, like NetBeans and iPython. Nevertheless, IWIB is renowned for its high or large code output, making debugging difficult when anything goes wrong. Boost. Python with pybind11 allow you to encapsulate C++ programs in C++ at a high level. Both are intended to be non-intrusive, making them excellent choices for exposing third-party libraries in Python. Pybind11 additionally provides specialized NumPy array support [22].

Cython is perhaps the most used Python wrapper for C/C++ programming [23]. Cython is a Python-like programming language as well as a compiler. The Cython programming language adds C type annotations to Python methods, variables, and class properties. It enables users to create Python script which calls pure C++ program back and forth. This produces optimized C program that can be immediately executed by the CPython interpreter when built. Cython extensions seem to users to be identical to another Python module. Almost all C++ capabilities, such as template structures and method overloading, are supported natively in the Cython language.

C++ Language Interface Foundation (CLIF) is used to create PyKaldi extension modules [24]. CLIF is a Google-developed open-source project that was just published. CLIF parses type information from a C++ header using LLVM and the Clang C++ compiler, which would further be utilized to validate the interface description for the same headers and create an extension segment. CLIF enables programmers to make on-the-fly changes to the C++ API. Rechristened classes, procedures, and methods; mapping operations to Python's mystic techniques (e.g., constructor to _init_ or destructor to _getitem_); managing template and functional overloading; and automatically creating setters and getters for subclass fields are all examples of this. CLIF was chosen for the reason that it enabled us to encapsulate Kaldi's code-base in a legible and simple manner. We can monitor Kaldi changes more effectively using CLIF since we don't have to make as many adjustments ourselves. Furthermore, CLIF's self-made code is far simpler to recite, comprehend, or change than SWIG or Cython's code. CLIF produced class containers don't have to be re-wrapped through composition in order to be immediately available in CPython, unlike in Cython. Without the requirement for specific procedural code or unwrapping, CLIF objects may very easily be inborn or simply given as parameters to bound roles. On the negative, CLIF is still a young project with few documentations and examples, as well as a nascent communal. CLIF's primary constraint mainly requires the program coding must be C++ v11 compatible and match Google C++ syntax, which required us to alter or expand the Kaldi codebase on occasion. CLIF produces a dummy doc-string, a text factual which further is being utilized to define the package in iPython, for every module, subclass, and function wrapped. We modified CLIF's default behavior to enable alternative docstrings to be provided within CLIF files to make documenting PyKaldi easier. The doc-strings discussed previously are tied to the relevant documentation sections of the related iPython segments, packages, and methods when they are supplied.

### *B. PyKaaldi Bundle*

PyKaaldi is designed in a modular manner, making it simple to maintain and expand without having to rewrite the whole software. The basic records are arranged in a folder-tree similar to the Kaldi structure. Every folder corresponds to a sub-package and solely includes wrapper programs for the Kaaldi module. The packaging code is as follows:

- C++ headers that define the slats for Kaldi program that isn't Google C++ compatible.
- The classes and methods to be wrapped, as well as their Python API, are described in CLIF C++ API specifications.
- Python packages are collections of interrelated CLIF set of methods that augment the basic CLIF packages to offer a better Python based API.

Though the containers produced by CLIF are usually sufficient for using Kaldi modules, PyKaldi often changes and expands the same in Python and in few other instances in C++, to improve the user interface. The remainder of this section contains useful examples of PyKaldi's new features.

#### *1) Matrix Bundle*

NumPy arrays, Python-Kaldi vector and matrix bundles are closely linked. Without duplicating the underlying memory buffers, they may be readily transformed to NumPy arrays and conversely. They additionally conform the NumPy array interface, allowing them to be included directly with functions that require NumPy arrays. Python-Kaldi vector and matrix bundle upkeep the usual Numpi cutting-edge indexing patterns merely by offloading the _setitem_ and _getitem_ NumPy functions, e.g.

```
v[k] = 2                  # set a vector item
m[i,j] = -1               # set a matrix item
m[:,j] = 0                # set a matrix column
m[:,::2] = m[:,1::2]      # set odd matrix columns
```

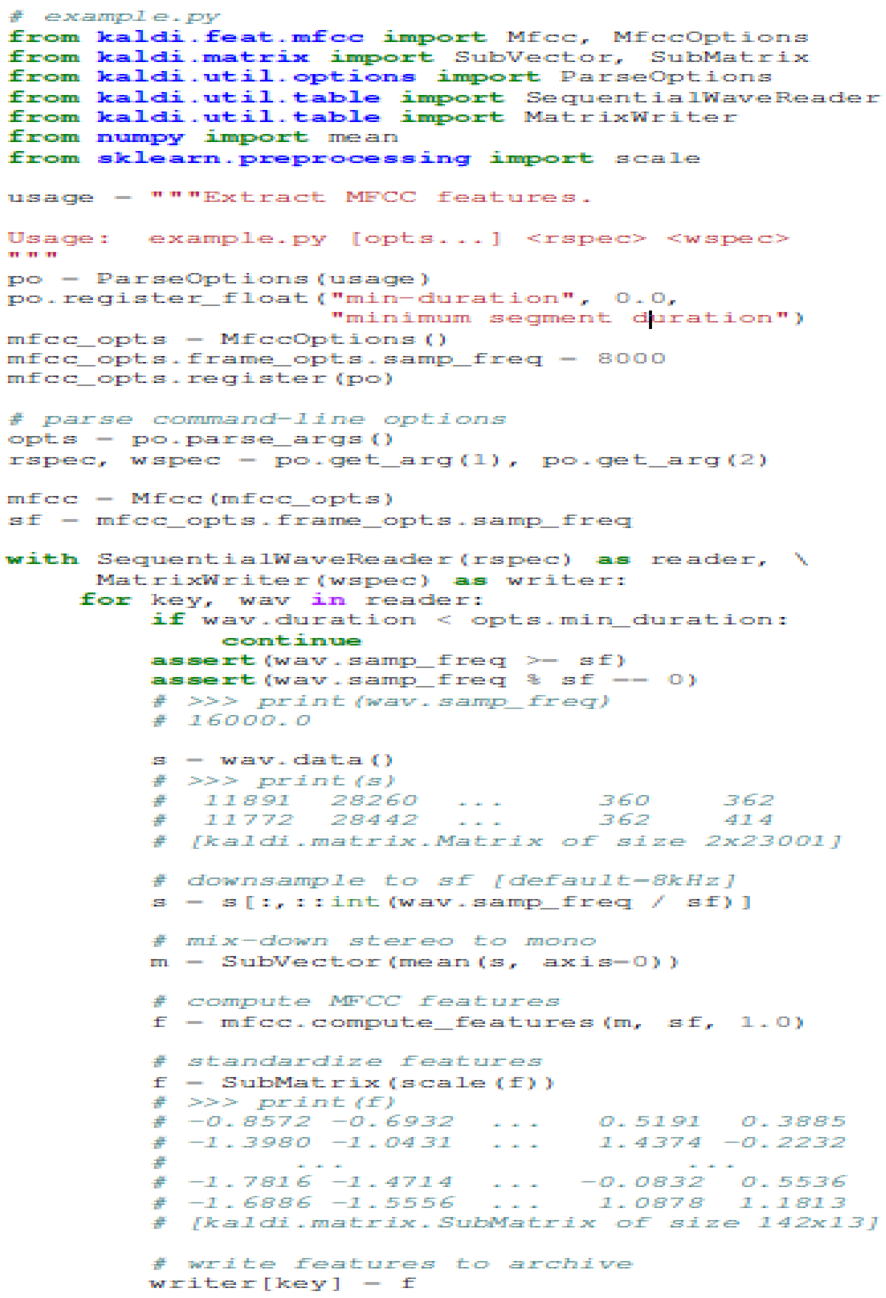
PyKaldi matrices and vectors may be made out of other array-like entities, and their instances can be made by copying elements from the source objects. When feasible, Sub-Vector and Sub-Matrix instances exchange data with the source objects that were used to create them. Only if the source object contains an array method that yields a copy, or if the parent object is a series, or when a copy is required to meet any of the other criteria, is a copy created (datatype, order, etc.). Sub-Vector and Sub-Matrix instances don't really own its memory buffers. Therefore, they maintain implicit relationships to objects with whom they share data to ensure that their storage channels are often not deallocated even though they are currently in scope.

#### *2) FST Package*

PyKaldi provides built-in functionality for basic FST types and functions including Kaldi lattices. The user-facing PyKaldi FST classes and operations API is fully written in Python, and it closely resembles the API provided by OpenFst's official Python wrapper. For visual analytics of FSTs, this provides interfaces with Graphviz and IPython. Unlike OpenFst's official Python wrapper, which is written in Cython, PyKaldi's OpenFst bindings are written in CLIF, ensuring that FST types are compatible with the remaining portion of the PyKaldi module. PyKaldi also does not cover OpenFst scripting API, which utilizes virtual dispatching, function registrations, and impact loads of shareable resources to offer a common interface used by FSTs of various semirings, unlike OpenFst's official Python wrapper. While this modification necessitates encapsulating every semiring variant of an OpenFst class or method template individually, it allows users to send PyKaldi FST features straight to the many Kaldi methods that take FST parameters.

### 3) *Error Handling*

Assertions are used extensively in the Kaldi codebase to verify the sanity of inputs and indeed the self-consistency of calculations. Regrettably, if a Kaldi assumption fails during execution, it causes an unfixable code abort, which would not be ideal during an ongoing Python session. Furthermore, while working collaboratively, all Kaldi problems, including assertion failures, display a stack trace, making it difficult to view the actual error message. We introduced additional methods to Kaldi to either disable or enable stack traces and indeed the abandon call in unsuccessful assumption processing to address these issues. Both are disabled by default in PyKaldi, but the user may enable them again. PyKaldi performs its own tests in Python in addition to Kaldi's sanity checks to ensure that the arguments given to Kaldi are of the right kinds and sizes.



```
# example.py
from kaldi.feat.mfcc import Mfcc, MfccOptions
from kaldi.matrix import SubVector, SubMatrix
from kaldi.util.options import ParseOptions
from kaldi.util.table import SequentialWaveReader
from kaldi.util.table import MatrixWriter
from numpy import mean
from sklearn.preprocessing import scale

usage = """Extract MFCC features.

Usage:  example.py [opts...] <rspec> <wspec>
"""
po = ParseOptions(usage)
po.register_float("min-duration", 0.0,
                  "minimum segment duration")
mfcc_opts = MfccOptions()
mfcc_opts.frame_opts.samp_freq = 8000
mfcc_opts.register(po)

# parse command-line options
opts = po.parse_args()
rspec, wspec = po.get_arg(1), po.get_arg(2)

mfcc = Mfcc(mfcc_opts)
sf = mfcc_opts.frame_opts.samp_freq

with SequentialWaveReader(rspec) as reader, \
     MatrixWriter(wspec) as writer:
    for key, wav in reader:
        if wav.duration < opts.min_duration:
            continue
        assert(wav.samp_freq >= sf)
        assert(wav.samp_freq % sf == 0)
        # >>> print(wav.samp_freq)
        # 16000.0

        s = wav.data()
        # >>> print(s)
        #  11891   28260   ...       360     362
        #  11772   28442   ...       362     414
        # [kaldi.matrix.Matrix of size 2x23001]

        # downsample to sf [default=8kHz]
        s = s[:,::int(wav.samp_freq / sf)]

        # mix-down stereo to mono
        m = SubVector(mean(s, axis=0))

        # compute MFCC features
        f = mfcc.compute_features(m, sf, 1.0)

        # standardize features
        f = SubMatrix(scale(f))
        # >>> print(f)
        # -0.8572 -0.6932   ...    0.5191   0.3885
        # -1.3980 -1.0431   ...    1.4374  -0.2232
        #          ...                ...
        # -1.7816 -1.4714   ...   -0.0832   0.5536
        # -1.6886 -1.5556   ...    1.0878   1.1813
        # [kaldi.matrix.SubMatrix of size 142x13]

        # write features to archive
        writer[key] = f
```

Figure. 4: PyKaldi is used to extract MFCC characteristics, is depicted in above picture

Fig. 4 shows a sample Python code for generating MFCC attributes using PyKaldi with NumPy and scikit-learn common tools. The program first configures the MFCC harvesting parameters, then uses an iterative over the input table, extracting and writing MFCC characteristics for each sound file. Before MFCC characteristics are retrieved, each sound file is compressed to 8KHz and blended down to mono. Before being printed down, raw MFCC characteristics are normalized by eliminating the mean and normalizing to random values. The PyKaldi option parsing API differs significantly from the Kaldi option parsing API. Type-specific enrolment techniques that take name, default value, and help text parameters, such as min-duration in the example, are used to register command-line parameters for the core program. A PyKaldi ParseOptions instance's parse _ARGS_ function produces a basic namespace object with the parsed option settings for the main script. Other alternative processed values are put straight into the relevant fields of related options objects, such as MFCC_OPTS there in example. In classic Kaldi form, read and write specifiers, or strings that define how the information should be read or written, are used to build input and output tables. The contextual management interface is implemented by PyKaldi table readers and writers, so they don't need to be stopped when utilised in a statement. For writing specified key value pairs, PyKaldi table writers offer a pseudo-dictionary approach. Because PyKaldi matrices use the NumPy array functionality, they may be given directly to methods that require Numpy array inputs, such as mean and scale. NumPy arrays may be readily changed back to Kaldi scalar and vector types by creating new Sub-Vector and Sub-Matrix instances that keep the associated storage buffers with the original arrays wherever feasible, i.e., no data is transferred until it's absolutely required.

## III. CONCLUSION

PyKaldi, an open-source and powerful scripting layer written in Python for Kaldi, was discussed. PyKaldi generates direct interfaces for the Kaldi C++ API using CLIF and expands the interfaces in Python to improve the client experience. PyKaldi actually implements a significant portion of the Kaldi C++ API at the stage of authoring. The next phase of the project, we think, will mostly concentrate on providing example configurations utilizing PyKaldi in conjunction with prominent Python modules, improving the API, and expanding the literature. We're optimistic that the Kaldi and Python audiences will appreciate PyKaldi and make a contribution to its future growth.

## REFERENCES

[1] Zalhan P, Stan A, Teodorescu L, Şaupe A, Duma M. A Kaldi-based ASR Solution for the Romanian Judicial System. 2016.

[2] Karan B, Sahoo J, Sahu PK. Automatic speech recognition based Odia system. In: 2015 International Conference on Microwave, Optical and Communication Engineering, ICMOCE 2015. 2016.

[3] Tahira M, Ather D, Saxena AK. Modeling and evaluation of heterogeneous networks for VANETs. In: Proceedings of the 2018 International Conference on System Modeling and Advancement in Research Trends, SMART 2018. 2018.

[4] Shukla S, Lakhmani A, Agarwal AK. A review on integrating ICT based education system in rural areas in India. In: Proceedings of the 5th International Conference on System Modeling and Advancement in Research Trends, SMART 2016. 2017.

[5] Tyagi S, Sexena A, Garg S. Secured high capacity Steganography using distribution technique with validity and reliability. In: Proceedings of the 5th International Conference on System Modeling and Advancement in Research Trends, SMART 2016. 2017.

[6] Sharma A, Sharma MK, Dwivedi RK. Novel approach of mining methods for social network sites. In: Proceedings of the 5th International Conference on System Modeling and Advancement in Research Trends, SMART 2016. 2017.

[7] Millman KJ, Aivazis M. Python for scientists and engineers. Computing in Science and Engineering. 2011.

[8] Chauhan N, Bhatt AK, Dwivedi RK, Belwal R. Accuracy testing of data classification using tensor flow a python framework in ANN designing. In: Proceedings of the 2018 International Conference on System Modeling and Advancement in Research Trends, SMART 2018. 2018.

[9] Khatri M, Kumar A. Stability Inspection of Isolated Hydro Power Plant with Cuttlefish Algorithm. In: 2020 International Conference on Decision Aid Sciences and Application, DASA 2020. 2020.

[10] Sharma K, Goswami L. RFID based Smart Railway Pantograph Control in a Different Phase of Power Line. In: Proceedings of the 2nd International Conference on Inventive Research in Computing Applications, ICIRCA 2020. 2020.

[11] Arabas S, Jarecka D, Jaruga A, Fijałkowski M. Formula translation in Blitz++, NumPy and modern Fortran: A case study of the language choice tradeoffs. Sci Program. 2014;

[12] Lavrijsen WTLP, Dutta A. High-performance python-C++ bindings with PyPy and cling. In: Proceedings of PyHPC 2016: 6th Workshop on Python for High-Performance and Scientific Computing - Held in conjunction with SC16: The International Conference for High Performance Computing, Networking, Storage and Analysis. 2017.

[13] Solanki MS, Goswami L, Sharma KP, Sikka R. Automatic Detection of Temples in consumer Images using histogram of Gradient. In: Proceedings of 2019 International Conference on Computational Intelligence and Knowledge Economy, ICCIKE 2019. 2019.

[14] Anand V. Photovoltaic actuated induction motor for driving electric vehicle. Int J Eng Adv Technol. 2019;

[15] Khanna R, Verma S, Biswas R, Singh JB. Implementation of branch delay in Superscalar processors by reducing branch penalties. In: 2010 IEEE 2nd International Advance Computing Conference, IACC 2010. 2010.

[16] Gupta H, Kumar S, Yadav D, Verma OP, Sharma TK, Ahn CW, et al. Data analytics and mathematical modeling for simulating the dynamics of COVID-19 epidemic—a case study of India. Electron. 2021;

[17] Gupta H, Varshney H, Sharma TK, Pachauri N, Verma OP. Comparative performance analysis of quantum machine learning with deep learning for diabetes prediction. Complex Intell Syst. 2021;

[18] Sharma TK. Enhanced butterfly optimization algorithm for reliability optimization problems. J Ambient Intell Humaniz Comput. 2021;

[19] Hirawat A, Taterh S, Sharma TK. A dynamic window-size based segmentation technique to detect driver entry and exit from a car. J King Saud Univ - Comput Inf Sci. 2021;

[20] PyKaldi [Internet]. [cited 2018 Aug 29]. Available from: https://github.com/pykaldi/pykaldi

[21] Can D, Martinez VR, Papadopoulos P, Narayanan SS. Pykaldi: A python wrapper for kaldi. In: ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings. 2018.

[22] pyBind11 [Internet]. [cited 2018 Aug 29]. Available from: https://github.com/pybind/pybind11

[23] Behnel S, Bradshaw R, Citro C, Dalcin L, Seljebotn DS, Smith K. Cython: The best of both worlds. Comput Sci Eng. 2011;

[24] Sutton A, Maletic JI. Emulating C++0x concepts. Sci Comput Program. 2013;