Quantum Audio

python 3.9+ | pypi v0.0.2 | API docs failing | License Apache 2.0 | DOI 10.1234/zenodo.123456 | QC Open in Colab

An open-source Python package for building Quantum Representations of Digital Audio using Qiskit circuits.

What is Quantum Audio ?

Quantum audio refers to the application of principles from quantum mechanics to the creation, processing, and analysis of sound or audio signals. Here, the information is processed using quantum bits, or qubits, instead of classical bits (0s and 1s).

The `quantumaudio` package provides fundamental operations for representing audio samples as Quantum States that can be processed on a Quantum computer (or a Simulator) and played back.

```
quantumaudio.encode(audio)   #returns a quantum circuit
quantumaudio.decode(circuit) #returns audio samples
```

Note

Quantum Audio represents Audio in terms of Quantum Circuits and does not require Quantum Memory for storage.

Installation

To install the Quantum Audio Package, run the following command in Terminal or Command Prompt:

```
pip install quantumaudio
```

Additional dependencies required for the demos provided in the GitHub repository can be installed using `pip` :

```
pip install "quantumaudio[demos]"
```

This includes Digital Audio Dependencies (*soundfile* and *librosa*) and an Interative notebook dependency (*ipywidgets*).

Overview

Modulation Schemes are essential methods for encoding Audio signals in both Analog (such as **FM**) and Digital (such as **PCM**) formats. The same is extended for Quantum Audio. The package contains different schemes to encode audio and necessary utilities.

The following subpackages can be accessed from `quantumaudio` :

- `schemes` : Quantum Audio Representation Methods
  - **QPAM**: Quantum Probability Amplitude Modulation (Original: Real-ket)
  - **SQPAM**: Single-Qubit Probability Amplitude Modulation (Original: FRQI)
  - **MSQPAM**: Multi-channel Single-Qubit Probability Amplitude Modulation (Original: PMQA)
  - **QSM**: Quantum State Modulation (Original: FRQA)
  - **MQSM**: Multi-channel Quantum State Modulation (Original: QRMA)
- `utils` : Common utilary functions for data processing, analysis, circuit preparation, etc.

Additionally, `tools` contain extension functions that support basic visual analysis and audio processing.

Usage

Using Schemes

Get started on creating Quantum Audio Representations with just a few lines of code.

```
# An instance of a scheme can be created using:
import quantumaudio
qpam = quantumaudio.load_scheme("qpam") # or directly access from quantumaudio.schemes.QPAM()

# Define an Input
original_data = quantumaudio.tools.test_signal() # for a random array of samples (range: -1.0 t

# Encoding
encoded_circuit = qpam.encode(original_data)

# ... optionally do some analysis or processing

# Decoding
decoded_data  = qpam.decode(encoded_circuit,shots=4000)
```

Using Functions

The core functions are also directly accessible without declaring a Scheme object. (Refer to Documentation for all the available functions)

```
circuit = quantumaudio.encode(data, scheme="qpam")
decoded_data = quantumaudio.decode(circuit)
```

Here, any remaining arguments can be passed as keywords e.g. `quantumaudio.encode(data, scheme="qsm", measure="False")` ,

Working with Digital Audio

For faster processing of longer arrays, the `stream` method is preferred.

```
quantumaudio.stream(data)
```

It wraps the functions provided in the module `quantumaudio.tools.stream` that help process large arrays as chunks for efficient handling. Examples of its usage can be found in the Demos provided in the repository.

Running on Native Backends

A Scheme's `decode()` method uses local AerSimulator as the default backend. Internally, the function calls `quantumaudio.utils.execute` method to perform `backend.run()` method. Any Qiskit compatible backend object can be specified by passing the `backend=` parameter to the `decode()` function.

Running on External Quantum Backends

The package allows flexible use of Quantum Hardware from different Providers as the execution of circuits can be done independently. Depending on the results, there are two ways to decode quantum audio:

- **Results Object:** If the result obtained follow the format of qiskit.result.Result or qiskit.primitives.PrimitiveResult,
  - The audio can be decoded with `scheme.decode_result(result_object)` method.
  - In this case, relevant metadata information is automatically extracted and applied at decoding. It can also be manually passed using `metadata=` parameter.
- **Counts Dictionary:** If the result is in form of a counts dictionary or qiskit.result.Counts object,
  - The audio can be decoded using `scheme.decode_counts(counts, metadata)` method.
  - The metadata dictionary can be accessed from the encoded circuit using `circuit.metadata` .

Using Custom Functions

The `decode` and `stream` operations can be configured with the following custom functions. They require few mandatory arguments followed by custom preceding keyword arguments (denoted as `**kwargs`).

- **Process Function**: The default process function of `stream()` simply encodes and decodes a chunk of data with default parameters. It can be overriden by passing a custom function to the `process_function=` parameter. The mandatory arguments for the custom process function are `data=` and `scheme=` ,

```
processed_data = process_function(data, scheme, **kwargs)
```

- **Execute Function**: The default execute function for `decode()` can be overriden by passing a custom function to the `execute_function=` parameter. The mandatory argument for the custom execute function is `circuit=` , (QPAM also expects `shots=` since it's a metadata)

```
result = execute_function(circuit, **kwargs)
```

Example: An optional execute function is included in the package which uses Sampler Primitive: `quantumaudio.utils.execute_with_sampler` that can be passed to the `decode()` method.

Version Information

Pre-release original version: `v0.0.2`

This project is derived from research output on Quantum Representations of Audio, carried by Interdisciplinary Centre for Computer Music Research (ICCMR), University of Plymouth, UK, namely:

- Itaboral, P.V., Miranda, E.R. (2022). Quantum Representations of Sound: From Mechanical Waves to Quantum Circuits. In: Miranda, E.R. (eds) Quantum Computer Music. Springer, Cham. https://doi.org/10.1007/978-3-031-13909-3_10
- Itaboral, P. V. (2022). Quantumaudio Module (Version 0.0.2) [Computer software]. https://github.com/iccmr-quantum/quantumaudio
- Itaboral, P. V. (2023) Towards Quantum Computing for Audio and Music Expression. Thesis. University of Plymouth. Available at: https://doi.org/10.24382/5119

Redevelopment: `v0.1.0`

This project has been completely re-developed and is now maintained by Moth Quantum.

- **New Architecture:**
  - This project has been restructured for better flexibility and scalability.
  - Instead of *QuantumAudio* Instances, the package begins at the level of *Scheme* Instances that perform encoding and decoding functions independent of the data.
- **Feature Updates:**
  - Introducing 2 Additional Schemes that can encode and decode Multi-channel Audio.
  - Supports Faster encoding and decoding of long audio files using Batch processing.
- **Dependency Change:**
  - Support for Qiskit is updated from `v0.22` to `v1.0+`
- **Improvements:**
  - Improved organisation of code for Readability and Modularity.
  - Key metadata information is preserved during the encoding operation, making the decoding process independent.
- **License Change:**
  - The License is updated from **MIT** to **Apache 2.0**

Citing

If you use this code or find it useful in your research, please consider citing: DOI

Copyright

Contact

If you have any questions or need further assistance, please feel free to contact Moth Quantum at hello@mothquantum.com

Documentation

Modules

Next