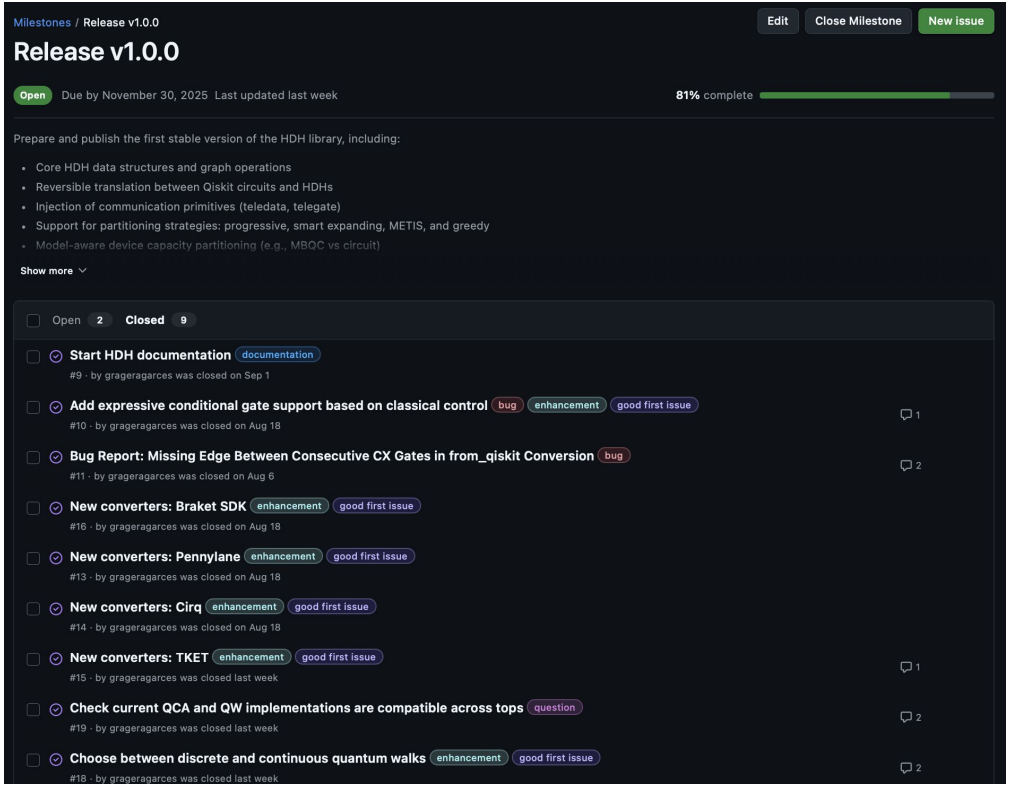
Unitary Foundation Midpoint Check-in

The HDH library | Maria Gragera Garces

What we did

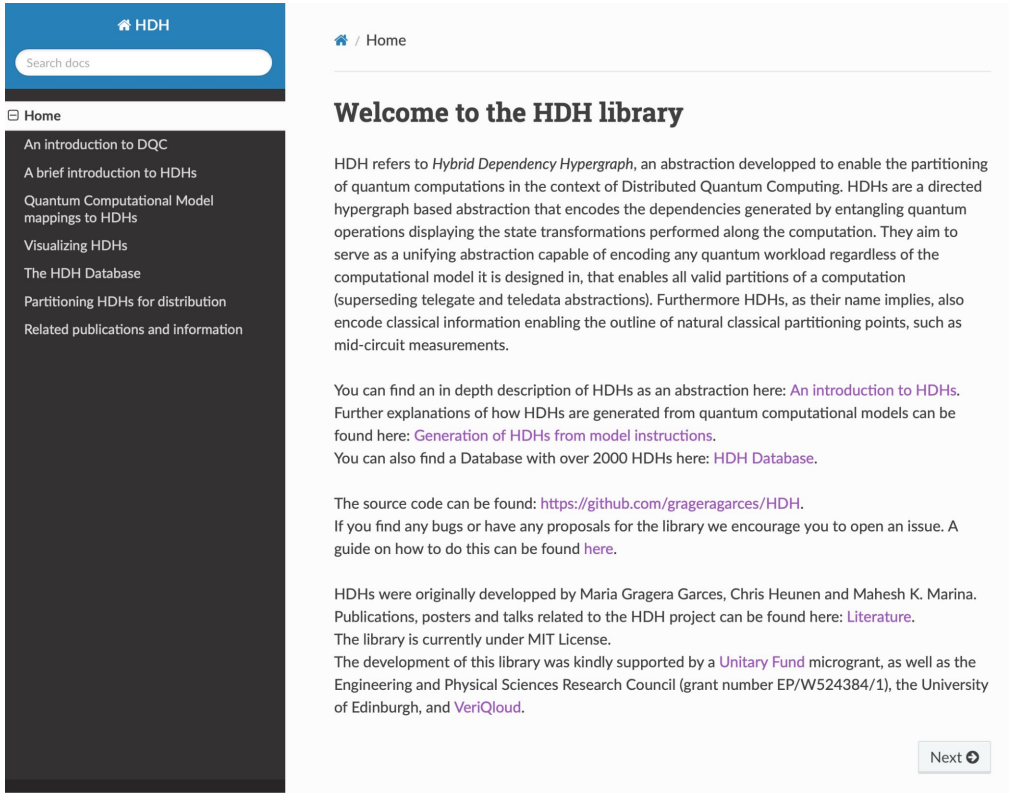
Closed over 80% of issues required for v1.0 release!



What we did

This included:

- **Public documentation**



What we did

This included:

- Public documentation
- **Compatibility with popular quantum SDKs**

```python
import hdh
import qiskit
from qiskit import QuantumCircuit
from hdh.converters.qiskit import from_qiskit
from hdh.visualize import plot_hdh
from hdh.passes.cut import compute_cut, cost, partition_sizes, compute_parallelism_by_time

# Qiskit circuit
qc = QuantumCircuit(3)
qc.h(0)
qc.cx(0, 1)
qc.ccx(1, 2, 0)
qc.measure_all()

hdh_graph = from_qiskit(qc) # Generate HDH
fig = plot_hdh(hdh) # Visualize HDH
```

Qiskit

OpenQASM

Amazon Braket

Currently unstable

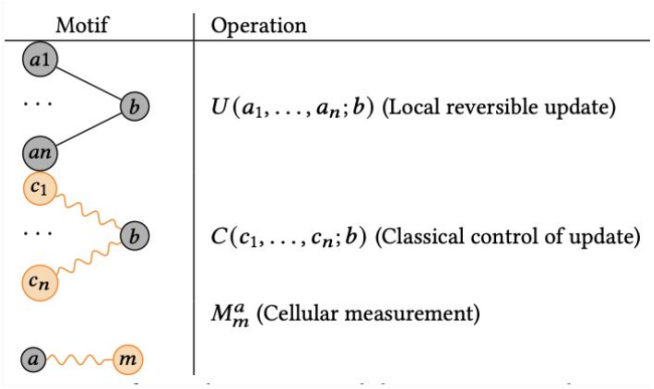
PENNYLANE

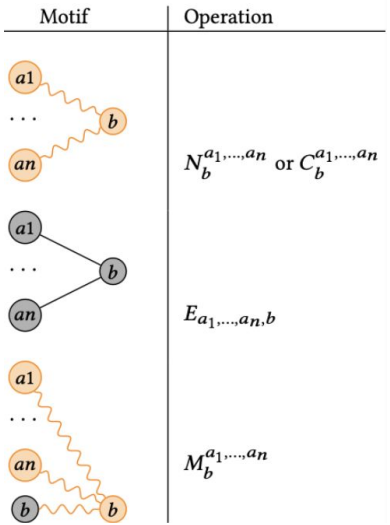
Cirq

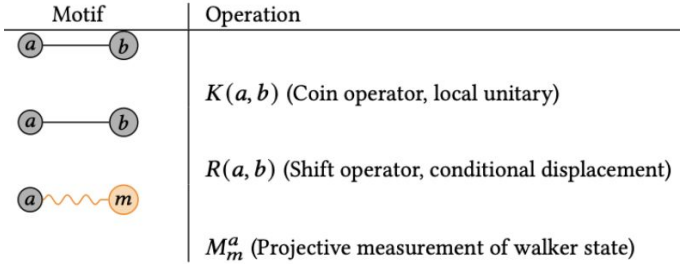
What we did

This included:

- Public documentation
- Compatibility with popular quantum SDKs
- **Further development of the theory**
  - **Models**







What we did

This included:

- Public documentation
- Compatibility with popular quantum SDKs
- **Further development of the theory**
  - Models
  - **Partitioners**

Ongoing implementation of state of the art partitioners

Partitioner

The main partitioning function is `compute_cut`, which implements a greedy, bin-filling algorithm. Here's how it works:

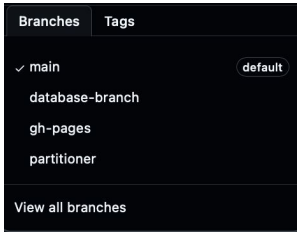
- **Node-level assignment:** The partitioner assigns individual nodes of the HDH graph to bins.
- **Qubit-based capacity:** While the assignment is at the node level, the capacity of each bin is determined by the number of *unique qubits* it contains.
- **Automatic sibling placement:** Once a node corresponding to a particular qubit is placed in a bin, all other nodes associated with that same qubit are automatically assigned to the same bin.
- **Ordering and selection:**
  - The algorithm first selects a representative node for each qubit, based on the weighted degree of the nodes.
  - These representatives are then sorted in descending order of their weighted degrees to prioritize high-connectivity qubits.
  - A beam search is used to select the best candidate nodes to place in each bin, considering the change in cut cost and a "frontier score" that measures how connected a node is to the nodes already in the bin.
- **Mop-up:** Any remaining unassigned nodes are distributed among the bins in a round-robin fashion, respecting the capacity constraints.

What we did

This included:

- Public documentation
- Compatibility with popular quantum SDKs
- **Further development of the theory**
  - Models
  - Partitioners
  - **Publication library v1.0**

What we did



This included:

- Public documentation
- Compatibility with popular quantum SDKs
- Further development of the theory
- **Development of the HDH database**

The HDH Database

To support reproducible evaluation and training of partitioning strategies, this library includes a database of pre-generated HDHs.
We aim for this resource to facilitate benchmarking across diverse workloads and enables the development of learning-based distribution agents.

Our goal is to extend the database with performance metrics of partitioning techniques for each workload. This will allow the community to build a data-driven understanding of which hypergraph partitioning methods perform best under different conditions. We encourage users to contribute results from their own partitioning methods when working with this database. Instructions for how to upload results can be found below.

The database is available in the `database-branch` of the repository. This separation ensures that users of the main library don't need to download unnecessary files.

The database currently contains:

- HDHs derived from the Munich Quantum Benchmarking Dataset.

The database is organized into two mirrored top-level folders:

- **Workloads**: contains the raw workload commands (currently QASM files).
- **HDHs**: contains the corresponding Hybrid Dependency Hypergraphs for each workload.

What we did

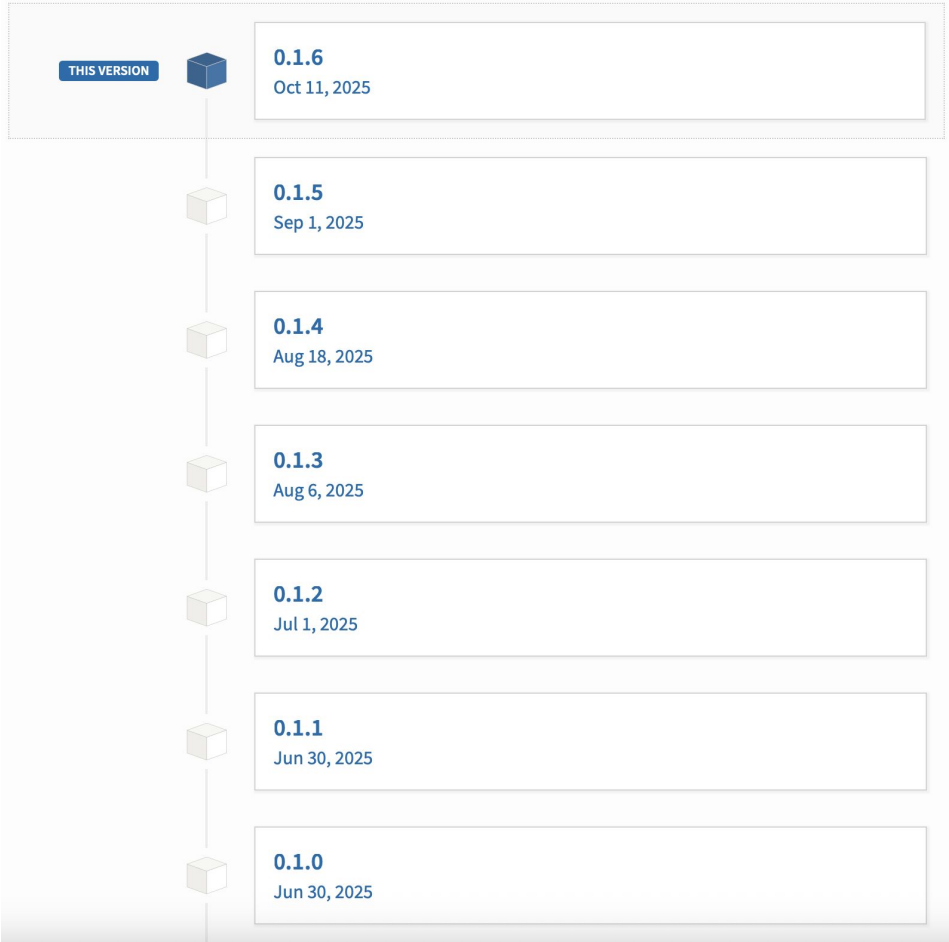
This included:

- Public documentation
- Compatibility with popular quantum SDKs
- Further development of the theory
- Development of the HDH database
- **Began supporting external contributions!**

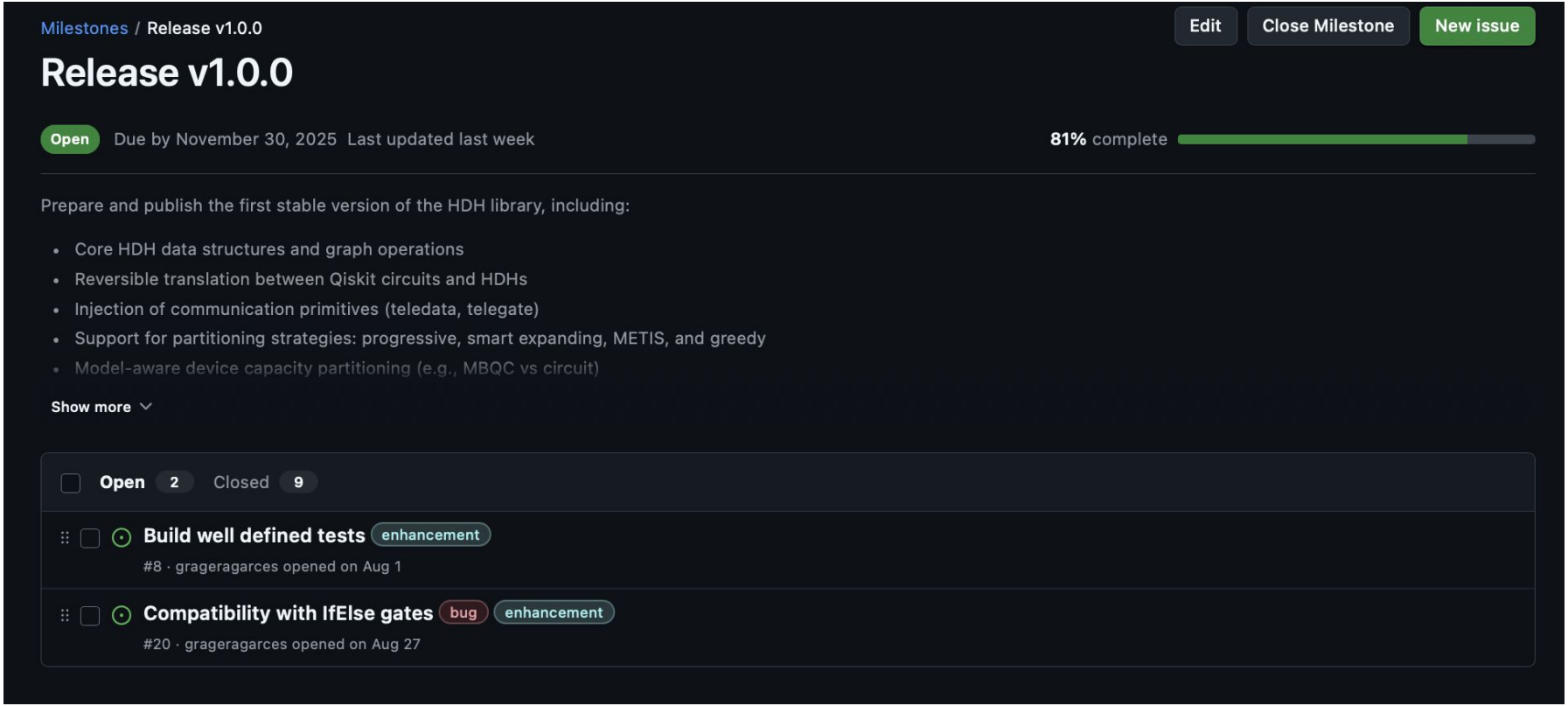
What we did

This included:

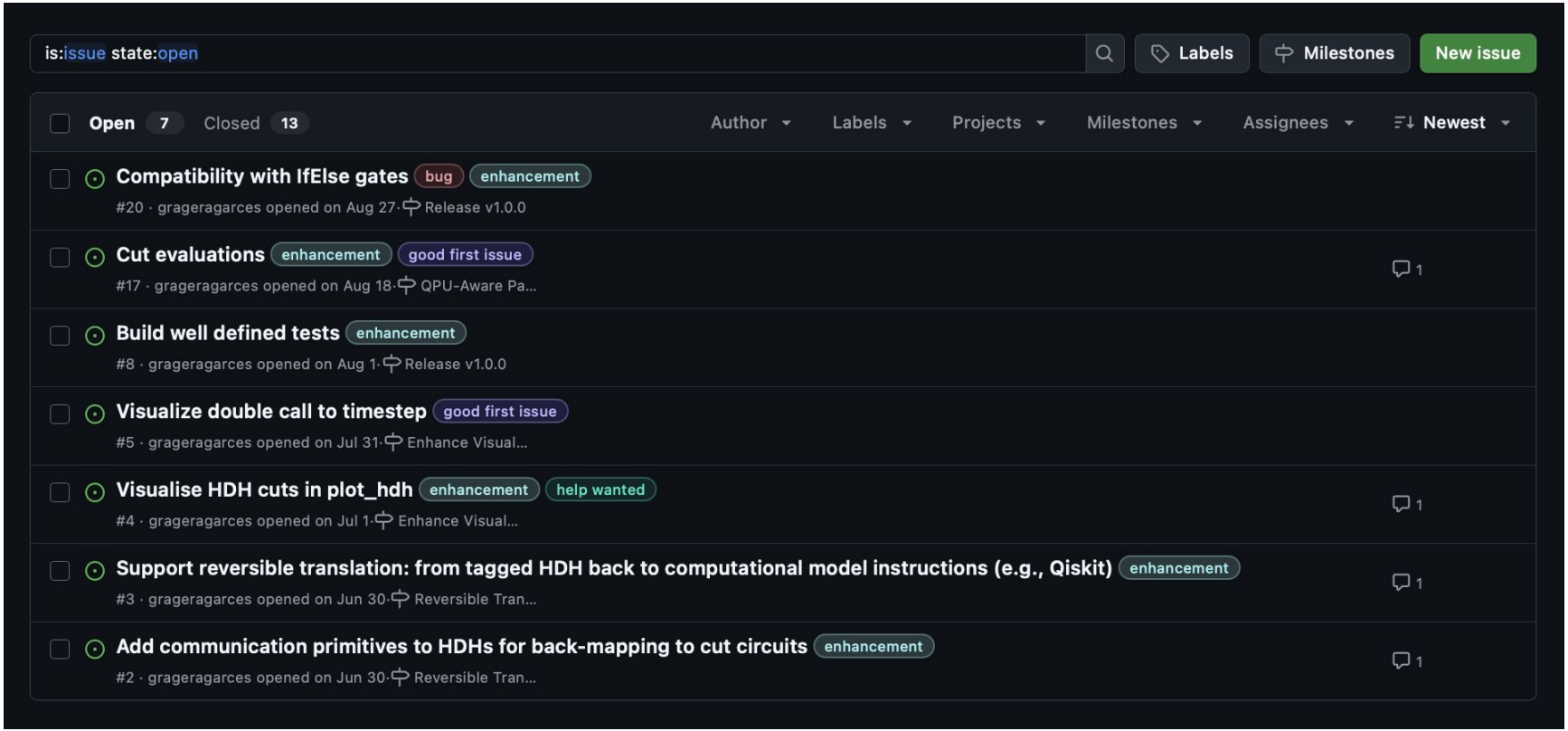
- Public documentation
- Compatibility with popular quantum SDKs
- Further development of the theory
- Development of the HDH database
- Began supporting external contributions!
- Went through major upgrades that included:
  - Visualization
  - Library tools
  - Ease of implementation



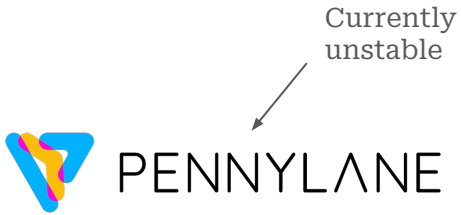
What we are doing next



What we are doing next



What we are doing next (not in issues yet)



Ongoing implementation of state of the art partitioners

Partitioner

The main partitioning function is `compute_cut`, which implements a greedy, bin-filling algorithm. Here's how it works:

- **Node-level assignment:** The partitioner assigns individual nodes of the HDH graph to bins.
- **Qubit-based capacity:** While the assignment is at the node level, the capacity of each bin is determined by the number of *unique qubits* it contains.
- **Automatic sibling placement:** Once a node corresponding to a particular qubit is placed in a bin, all other nodes associated with that same qubit are automatically assigned to the same bin.
- **Ordering and selection:**
  - The algorithm first selects a representative node for each qubit, based on the weighted degree of the nodes.
  - These representatives are then sorted in descending order of their weighted degrees to prioritize high-connectivity qubits.
  - A beam search is used to select the best candidate nodes to place in each bin, considering the change in cut cost and a "frontier score" that measures how connected a node is to the nodes already in the bin.
- **Mop-up:** Any remaining unassigned nodes are distributed among the bins in a round-robin fashion, respecting the capacity constraints.

What we are doing next (not in issues yet)

I’m not a big fan of the current database structuring:

1) Will re-arrange data for ease of use and access
2) Will add partitioning technique implementation data in more detail (currently cut descriptions are ambiguous)

The HDH Database

To support reproducible evaluation and training of partitioning strategies, this library includes a database of pre-generated HDHs.
We aim for this resource to facilitate benchmarking across diverse workloads and enables the development of learning-based distribution agents.

Our goal is to extend the database with performance metrics of partitioning techniques for each workload. This will allow the community to build a data-driven understanding of which hypergraph partitioning methods perform best under different conditions. We encourage users to contribute results from their own partitioning methods when working with this database. Instructions for how to upload results can be found below.

The database is available in the `database-branch` of the repository. This separation ensures that users of the main library don’t need to download unnecessary files.

The database currently contains:

- HDHs derived from the Munich Quantum Benchmarking Dataset.

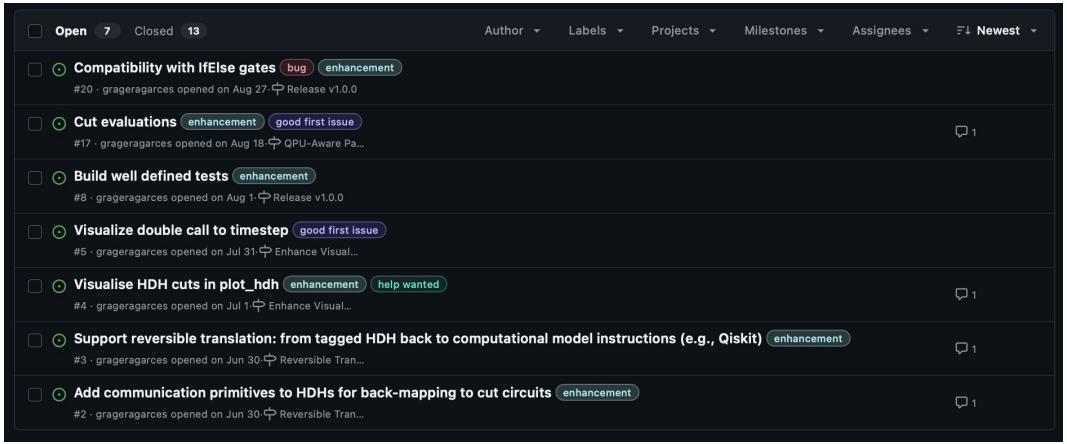
The database is organized into two mirrored top-level folders:

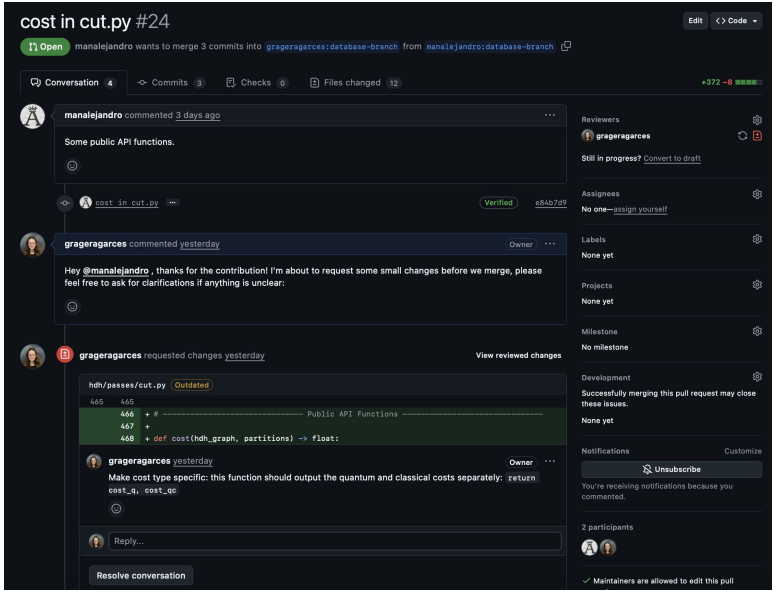
- **Workloads**: contains the raw workload commands (currently QASM files).
- **HDHs**: contains the corresponding Hybrid Dependency Hypergraphs for each workload.

V 1.0 -> submission

What we need help with

Any currently open issue!





Some may be more involved than others, thus why I try to tag them appropriately.

If you'd like to participate but need more guidance make a comment asking for clarifications!

Thanks for listening!

The HDH library | Maria Gragera Garces