

Unitary Foundation Midpoint Check-in

The HDH library | Maria Gragera Garces

What we did

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

Milestones / Release v1.0.0 Edit Close Milestone New issue

Release v1.0.0

Open Due by November 30, 2025 Last updated last week 81% complete

Prepare and publish the first stable version of the HDH library, including:

- Core HDH data structures and graph operations
- Reversible translation between Qiskit circuits and HDHs
- Injection of communication primitives (teledata, delegate)
- Support for partitioning strategies: progressive, smart expanding, METIS, and greedy
- Model-aware device capacity partitioning (e.g., MBQC vs circuit)

Show more ▾

Open	Closed
2	9

Start HDH documentation documentation #9 · by gragergarces was closed on Sep 1

Add expressive conditional gate support based on classical control bug enhancement good first issue #10 · by gragergarces was closed on Aug 18 1 comment

Bug Report: Missing Edge Between Consecutive CX Gates in from_qiskit Conversion bug #11 · by gragergarces was closed on Aug 6 2 comments

New converters: Braket SDK enhancement good first issue #16 · by gragergarces was closed on Aug 18

New converters: PennyLane enhancement good first issue #13 · by gragergarces was closed on Aug 18

New converters: Cirq enhancement good first issue #14 · by gragergarces was closed on Aug 18

New converters: TKET enhancement good first issue #15 · by gragergarces was closed last week 1 comment

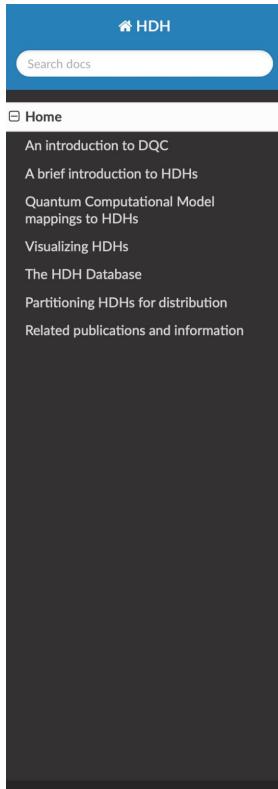
Check current QCA and QW implementations are compatible across tops question #19 · by gragergarces was closed last week 2 comments

Choose between discrete and continuous quantum walks enhancement good first issue #18 · by gragergarces was closed last week 2 comments

What we did

This included:

- Public documentation



/ Home

Welcome to the HDH library

HDH refers to *Hybrid Dependency Hypergraph*, an abstraction developed to enable the partitioning of quantum computations in the context of Distributed Quantum Computing. HDHs are a directed hypergraph based abstraction that encodes the dependencies generated by entangling quantum operations displaying the state transformations performed along the computation. They aim to serve as a unifying abstraction capable of encoding any quantum workload regardless of the computational model it is designed in, that enables all valid partitions of a computation (superseding telegate and teledata abstractions). Furthermore HDHs, as their name implies, also encode classical information enabling the outline of natural classical partitioning points, such as mid-circuit measurements.

You can find an in depth description of HDHs as an abstraction here: [An introduction to HDHs](#). Further explanations of how HDHs are generated from quantum computational models can be found here: [Generation of HDHs from model instructions](#).

You can also find a Database with over 2000 HDHs here: [HDH Database](#).

The source code can be found: <https://github.com/grageragarces/HDH>.

If you find any bugs or have any proposals for the library we encourage you to open an issue. A guide on how to do this can be found [here](#).

HDHs were originally developed by Maria Gragera Garces, Chris Heunen and Mahesh K. Marina. Publications, posters and talks related to the HDH project can be found here: [Literature](#).

The library is currently under MIT License.

The development of this library was kindly supported by a [Unitary Fund](#) microgrant, as well as the Engineering and Physical Sciences Research Council (grant number EP/W524384/1), the University of Edinburgh, and [VeriQloud](#).

Next

What we did

This included:

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

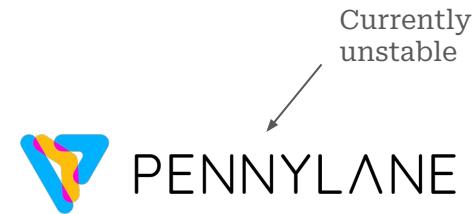
```
import hdh
import qiskit
from qiskit import QuantumCircuit
from hdh.converters.qiskit import from_qiskit
from hdh.visualize import plot_hdh
from hdh.passes.cut import compute_cut, cost, partition_sizes, compute_parallelism_by_time

# 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
```



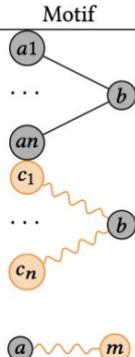
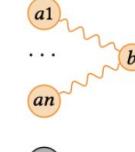
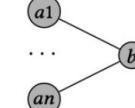
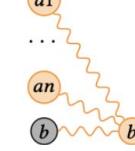
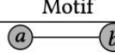
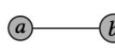
OpenQASM



What we did

This included:

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

Motif	Operation	Motif	Operation
	$U(a_1, \dots, a_n; b)$ (Local reversible update) $C(c_1, \dots, c_n; b)$ (Classical control of update) M_m^a (Cellular measurement)	 	$N_b^{a_1, \dots, a_n}$ or $C_b^{a_1, \dots, a_n}$
	$E_{a_1, \dots, a_n, b}$		$M_b^{a_1, \dots, a_n}$
Motif	Operation	Motif	Operation
  	$K(a, b)$ (Coin operator, local unitary) $R(a, b)$ (Shift operator, conditional displacement) M_m^a (Projective measurement of walker state)		

What we did

This included:

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

Ongoing implementation of state of the art partitioners

What we did

This included:

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

The screenshot shows the homepage of The Journal of Open Source Software (JOSS). At the top, there is a navigation bar with links for About, Papers, Docs, Blog, Submit, My Profile, and Log out, along with social media icons for GitHub and RSS feed. The main title "About the Journal of Open Source Software" is centered above a brief description: "The Journal of Open Source Software is a developer friendly, open access journal for research software packages." Below this, there are several sections with headings and descriptive text. A sidebar on the right lists various "About" topics such as Submission scope, Editorial Board, Topic Editors, Editors Emeritus, Contact JOSS, Code of Conduct, Ethics Guidelines, Cost and Sustainability Model, and Content Licensing & Open Access. At the bottom of the sidebar is a "Author information" button.

About the Journal of Open Source Software

The Journal of Open Source Software is a developer friendly, open access journal for research software packages.

What exactly do you mean by 'journal'

The Journal of Open Source Software (JOSS) is an academic journal (ISSN 2475-9066) with a formal peer review process that is designed to *improve the quality of the software submitted*. Upon acceptance into JOSS, a Crossref DOI is minted and we list your paper on the JOSS website.

Don't we have enough journals already?

Perhaps, and in a perfect world we'd rather papers about software weren't necessary but we recognize that for most researchers, papers and not software are the currency of academic research and that citations are required for a good career.

We built this journal because we believe that after you've done the hard work of writing great software, it shouldn't take weeks and months to write a paper about your work.

You said *developer friendly*, what do you mean?

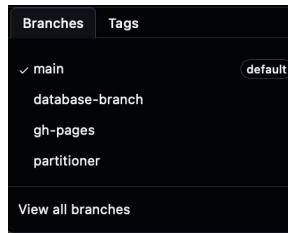
We have a simple submission workflow and [extensive documentation](#) to help you prepare your submission. If your software is already well documented then paper preparation should take no more than an hour.

You can read more about our motivations to build JOSS in our [announcement blog post](#).

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:

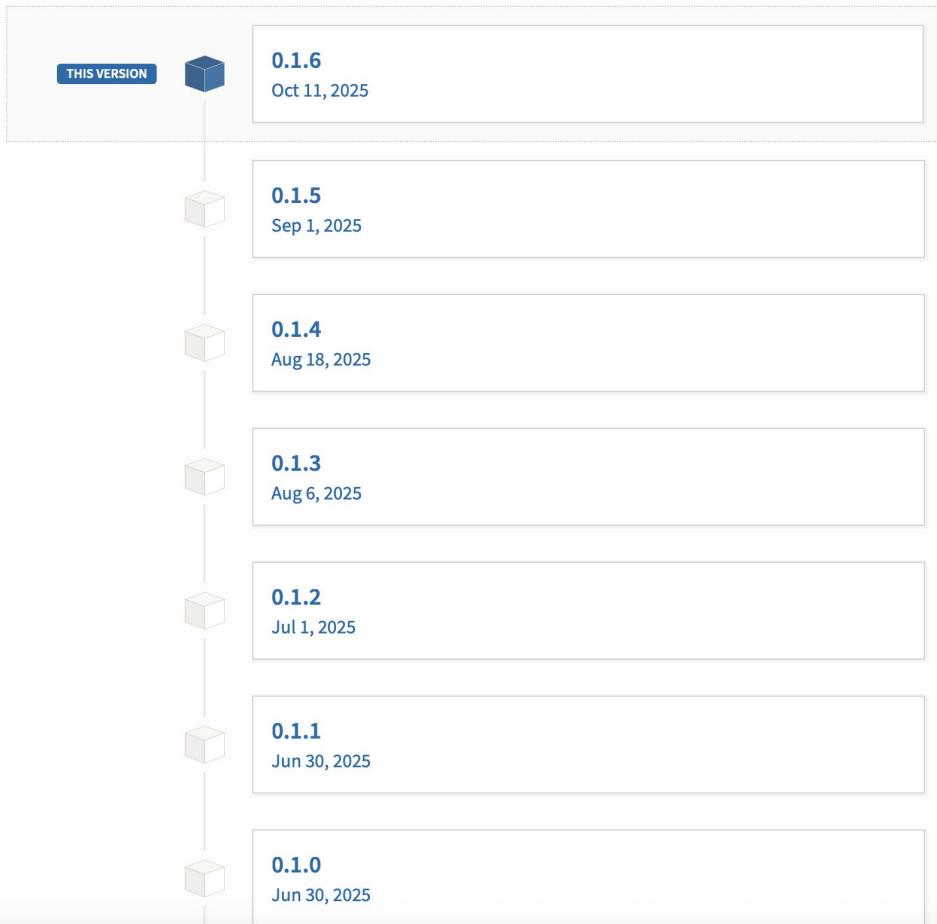
- Public documentation
- Compatibility with popular quantum SDKs
- Further development of the theory
- Development of the HDH database
- **Began supporting external contributions!**



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

Milestones / Release v1.0.0

Release v1.0.0

Open Due by November 30, 2025 Last updated last week **81% complete** 

Prepare and publish the first stable version of the HDH library, including:

- Core HDH data structures and graph operations
- Reversible translation between Qiskit circuits and HDHs
- Injection of communication primitives (teldata, delegate)
- Support for partitioning strategies: progressive, smart expanding, METIS, and greedy
- Model-aware device capacity partitioning (e.g., MBQC vs circuit)

Show more ▾

Open 2 **Closed** 9

⋮ **Build well defined tests** enhancement #8 · grageragarces opened on Aug 1

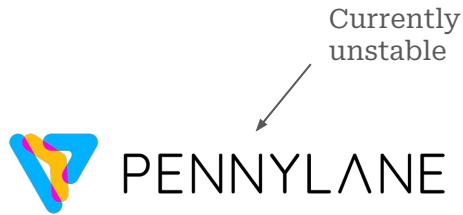
⋮ **Compatibility with IfElse gates** bug enhancement #20 · grageragarces opened on Aug 27

What we are doing next

The screenshot shows a GitHub search results page with the query `is:issue state:open`. The results are sorted by **Newest**. There are 7 open issues listed:

- Compatibility with IfElse gates** (`bug enhancement`)
#20 · gragergarces opened on Aug 27 · [Release v1.0.0](#)
- Cut evaluations** (`enhancement good first issue`)
#17 · gragergarces opened on Aug 18 · [QPU-Aware Pa...](#) 1 comment
- Build well defined tests** (`enhancement`)
#8 · gragergarces opened on Aug 1 · [Release v1.0.0](#)
- Visualize double call to timestep** (`good first issue`)
#5 · gragergarces opened on Jul 31 · [Enhance Visual...](#)
- Visualise HDH cuts in plot_hdh** (`enhancement help wanted`)
#4 · gragergarces opened on Jul 1 · [Enhance Visual...](#) 1 comment
- Support reversible translation: from tagged HDH back to computational model instructions (e.g., Qiskit)** (`enhancement`)
#3 · gragergarces opened on Jun 30 · [Reversible Tran...](#) 1 comment
- Add communication primitives to HDHs for back-mapping to cut circuits** (`enhancement`)
#2 · gragergarces opened on Jun 30 · [Reversible Tran...](#) 1 comment

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

The screenshot shows the homepage of The Journal of Open Source Software. At the top, there's a navigation bar with links for About, Papers, Docs, Blog, Submit, My Profile, Log out, and a search icon. Below the navigation is a section titled "About the Journal of Open Source Software". It includes a brief description: "The Journal of Open Source Software is a developer friendly, open access journal for research software packages." To the right of this main content area is a sidebar with various links under the heading "About", such as Submission scope, Editorial Board, Topic Editors, Editors Emeritus, Contact JOSS, Code of Conduct, Ethics Guidelines, Cost and Sustainability Model, Content Licensing & Open Access, and Author Information.

What we need help with

Any currently open issue!

A screenshot of a GitHub search results page. The search bar at the top shows the query "cost in cut.py". The results list several open issues:

- Compatibility with ifElse gates (bug, enhancement)
- Cut evaluations (enhancement, good first issue)
- Build well defined tests (enhancement)
- Visualize double call to timestep (good first issue)
- Visualise HDH cuts in plot_hdh (enhancement, help wanted)
- Support reversible translation: from tagged HDH back to computational model instructions (e.g., Qiskit) (enhancement)
- Add communication primitives to HDHs for back-mapping to cut circuits (enhancement)

Each issue includes a brief description, the number of comments (e.g., 1), and the date it was opened.

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!

A screenshot of a GitHub pull request conversation. The pull request is titled "cost in cut.py #24" and is from manalejandro into gragergarces/database-branch. The conversation shows:

- manalejandro commented 3 days ago: Some public API functions.
- gragergarces commented yesterday: Hey @manalejandro, thanks for the contribution! I'm about to request some small changes before we merge, please feel free to ask for clarifications if anything is unclear.
- gragergarces requested changes yesterday: hdh/passes/cut.py Outdated. A diff shows the addition of a "cost" function.

```
465 466
466 + # -----
467 +
468 + def cost(hd_graph, partitions) -> float:
```
- gragergarces yesterday: Make cost type specific: this function should output the quantum and classical costs separately: return cost_q, cost qc

The right sidebar shows the pull request details: 372 reviews, 84b7d9 status, 1 assignee (No one—assign yourself), 0 reviewers (gragergarces), 0 labels (None yet), 0 projects (None yet), 0 milestones (None), 0 development (Successfully merging this pull request may close these issues. None yet), 0 notifications (Customize), and 2 participants (A, B).

Thanks for listening!

The HDH library | Maria Gragera Garces