XQuantum

How can we generate a good explanation of a quantum program?

For software engineers or students, who might be unfamiliar with QC

XQuantum

Evaluating Explanations for Software Patches Generated by Large Language Models

Dominik Sobania[1] [0000−0001−8873−7143], Alina Geiger[1] [0009−0002−3413−283X], James Callan[2] [0000−0002−5692−6203], Alexander Brownlee[3] [0000−0003−2892−5059], Carol Hanna[2] [0009−0009−7386−1622], Rebecca Moussa[2] [0000−0001−9123−6008], Mar Zamorano López[2] [0000−0002−8872−4876], Justyna Petke[2] [0000−0002−7833−6044], and Federica Sarro[2] [0000−0002−9146−442X]

SSBSE 2023


XQuantum
Quantum Algorithms
Prompts
LLM
Generated Textual Explanation
Manual Analysis


XQuantum
Quantum Algorithms
• Qasm
• Qiskit
• Circuit
Prompts
LLM
Generated Textual Explanation
Manual Analysis


XQuantum
Quantum Algorithms
• Qasm
• Qiskit
• Circuit
Prompts
3 Levels
of detail
LLM
Generated Textual
Explanation
Manual Analysis


XQuantum
Quantum Algorithms
Qasm
Qiskit
Circuit
Prompts
3 Levels of detail
LLM
GPT 3.5
Tiny Llama
Llama2
Generated Textual Explanation
Manual Analysis

XQuantum

To understand this piece of code, let's start by defining what a QuantumRegister (QR) and QuantumCircuit (QC) are. A QR is a collection of qubits that can be manipulated in parallel using different gates. It consists of a list of qubit indices (`q`), along with their corresponding gate operations (`c`).

In the above code, `q` is an array of four qubits (`q[0]`, `q[1]`, `q[2]`, and `q[3]`). Each gate operation (`c`) is a QC instruction that applies one or more gates to these qubits. The final step in this circuit is the `cu1` operation, which applies a controlled-U gate on qubit `1` (which is at index `0` in the original list) to qubit `3`.

To execute the circuit, we first initialize our QR with the desired values for each qubit (`qc.h(q[0])`), then prepare an Oracle circuit (`qc.x(q[0])`, `qc.x(q[1])`, `qc.x(q[2])`, and `qc.x(q[3]))`. Next, we apply the Controlled-U gate to `c`, followed by the final CX gate (`qc.cx(q[0], q[3])`).

Finally, we execute this entire circuit using a QuantumRegister, passing in `pi/4` and `pi/4` as arguments to the `cu1()` method. Note that the final step of executing the circuit is the `cx()` operation, which applies a controlled CX gate on qubit `0`, followed by a controlled CX gate on qubit `3`.

XQuantum
Quantum Algorithms
• Qasm
• Qiskit
• Circuit
Prompts
3 Levels of detail
LLM
• GPT 3.5
• Tiny Llama
• Llama2
Generated Textual Explanation
Manual Analysis
To understand this piece of code, let's start by defining what a QuantumRegister (QR) and QuantumCircuit (QC) are. A QR is a collection of qubits that can be manipulated in parallel using different gates. It consists of a list of qubit indices (`q`), along with their corresponding gate operations (`c`).
In the above code, `q` is an array of four qubits (`q[0]`, `q[1]`, `q[2]`, and `q[3]`). Each gate operation (`c`) is a QC instruction that applies one or more gates to these qubits. The final step in this circuit is the `cu1` operation, which applies a controlled-U gate on qubit `1` (which is at index `0` in the original list) to qubit `3`.
To execute the circuit, we first initialize our QR with the desired values for each qubit (`qc.h(q[0])`), then prepare an Oracle circuit (`qc.x(q[0])`, `qc.x(q[1])`, `qc.x(q[2])`, and `qc.x(q[3]))`. Next, we apply the Controlled-U gate to `c`, followed by the final CX gate (`qc.cx(q[0], q[3])`).
Finally, we execute this entire circuit using a QuantumRegister, passing in `pi/4` and `pi/4` as arguments to the `cu1()` method. Note that the final step of executing the circuit is the `cx()` operation, which applies a controlled CX gate on qubit `0`, followed by a controlled CX gate on qubit `3`.

XQuantum


Quantum Algorithms
• Qasm
• Qiskit
• Circuit
Prompts
3 Levels of detail
LLM
• GPT 3.5
• Tiny Llama
• Llama2
Generated Textual Explanation
To understand this piece of code, let's start by defining what a QuantumRegister (QR) and QuantumCircuit (QC) are. A QR is a collection of qubits that can be manipulated in parallel using different gates. It consists of a list of qubit indices (`q`), along with their corresponding gate operations (`c`).
In the above code, `q` is an array of four qubits (`q[0]`, `q[1]`, `q[2]`, and `q[3]`). Each gate operation (`c`) is a QC instruction that applies one or more gates to these qubits. The final step in this circuit is the `cu1` operation, which applies a controlled-U gate on qubit `1` (which is at index `0` in the original list) to qubit `3`.
To execute the circuit, we first initialize our QR with the desired values for each qubit (`qc.h(q[0])`), then prepare an Oracle circuit (`qc.x(q[0])`, `qc.x(q[1])`, `qc.x(q[2])`, and `qc.x(q[3])`)`. Next, we apply the Controlled-U gate to `c`, followed by the final CX gate (`qc.cx(q[0], q[3])`).
Finally, we execute this entire circuit using a QuantumRegister, passing in `pi/4` and `pi/4` as arguments to the `cu1()` method. Note that the final step of executing the circuit is the `cx()` operation, which applies a controlled CX gate on qubit `0`, followed by a controlled CX gate on qubit `3`.
Manual Analysis
Criteria Grid
• Correctness
• Completeness
• Complexity
• Continuity

XQuantum

- Have cross-validation
- Don’t use a grid but a simple comparison
- Expert and less experts
- Give guidance
- How much knowledge of QC do they have?