QuantumNFT Explained

Abstract

Introduction

QuantumNFT.xyz (QuantumNFT) is yet another quantum-readiness project by pQCee.com. It seeks to bring together a whole new generation of quantum-capable programmers as the backbone to usher in the next computing paradigm --- quantum computers.

After over 40 years of research and development by physicists around the world, commercially-useful quantum computers are expected to become mainstream within the next 5 years. Its potential applications range from the discovery of new chemical compounds to more accurate modelling and optimization of highly complex systems to faster processing and analysis of huge amounts of data. Market estimates on the use of quantum computers exceed US$4 billion in 2028, growing at a massive 38% CAGR. These studies point to more than just upgrades to the computing hardware, but a wider expansion in system requirements to include redesigning existing workflows and standards, new expertise in programming and maintaining quantum software, and the integration of quantum computers into everyday life.

The question we ask is if the broader technical community is prepared for this change? If all of the Fortune 1000 companies want to build in-house quantum teams to jumpstart their next leap in productivity, are there enough people to hire? Do we know how to find them?

Problem we are solving

When we take a count of software developers in the world, estimates from various sources found on Google.com all exceed 20 million developers. If we dive deeper into a niche community such as the blockchain (or Web3), this number hovers at around 30,000 active developers. But a search on the number of physicists working on quantum computers is likely to yield less than 3,000 worldwide. This startling shortfall of quantum-capable developers means that we are going to see a repeat of the 1970s where regular computers, despite heavy investments, did not yield the promised productivity growth till over a decade later.

But bridging this gap is not a matter of conducting more quantum classes. At present, despite the public availability of small-scale quantum computing resources on the cloud, the users have been mainly physicists who have used quantum computers for academic publication. The level of interest amongst undergraduates and enthusiasts who have attended quantum 101 courses has quickly waned due to the lack of applicability of quantum computers in their day-to-day lives. Which leads us to the problem --- the lack of a viable use-case or “killer-app” for small-scale quantum computers.

There has been no lack of efforts by academic and industry researchers to conduct experiments to demonstrate the usefulness or advantage of quantum computers, but most of these have either been inconsistent (i.e. results are not guaranteed), disproved (i.e. regular computers may perform better), or impractical (i.e. experiments are not general purpose enough). This is understandable, given the noisy and relatively-small state of quantum computers today. Instead of trying to outperform regular computers, our approach is to find a use-case that can complement existing applications to achieve the goal of building a community of at least 10,000 quantum developers.

Solution Design

To build the community, we have chosen to target following two segments :

- Computer Science / Engineering undergraduates. We expect these undergraduates to possess a requisite level of programming logic and knowledge, coupled with the drive to distinguish themselves when they enter into the labour market which has seen significant layoffs in recent months.
- Web3 developers. Many of these web3 developers are self-starters and have persisted in adapting to web3 despite the myriad of ever-changing programming languages, frameworks and rules. Since we expect quantum programming to also undergo similar changes, the resilience of web3 developers will feature well here.

The NFT (Non-Fungible Token) has been commonly used as a mechanism to bind communities together. Ownership of a NFT within a particular collection imbues some level of recognition that the owner is part of an exclusive community. Whether the ownership can be transferred/sold or imposing limits to the quantity of NFTs that are in circulation then defines the nature of the community. Since NFTs are also familiar to Web3 developers, we have chosen to use NFTs, named QuantumNFT, as the mechanism to build and maintain the quantum-developer community.

The general idea of QuantumNFT works as follows:

- A quantum developer writes different quantum programs.
- Each quantum program can be minted into a NFT as a representation of that program. This NFT should be deterministically displayed as an image such that the NFT will always yield the same image.
- Different developers who have written the same quantum program should mint similar NFTs that will display the same image. The only difference should be a serial number to differentiate the order of minting.
- A quantum developer should not be allowed to mint NFTs using the same program more than once.
- A quantum developer who has minted many different NFTs will be able to demonstrate higher capability in quantum programming as compared to another developer who has minted less NFTs.
- A quantum developer who has minted specific NFTs which other developers are unable to mint will be able demonstrate mastery in quantum programming.

The characteristics of the QuantumNFT collection are:

- NFT identification policy: Each NFT is uniquely identified by a TokenID which should include identification information about the quantum program, as well as a unique serial number.
- Number of NFTs in circulation: Unlimited
- Minting: To be done by quantum developer.
- Selling/Transfer: No selling or transfer allowed.

Implementation

The walk-through of how a quantum developer can mint a QuantumNFT is shown in Figure 1.

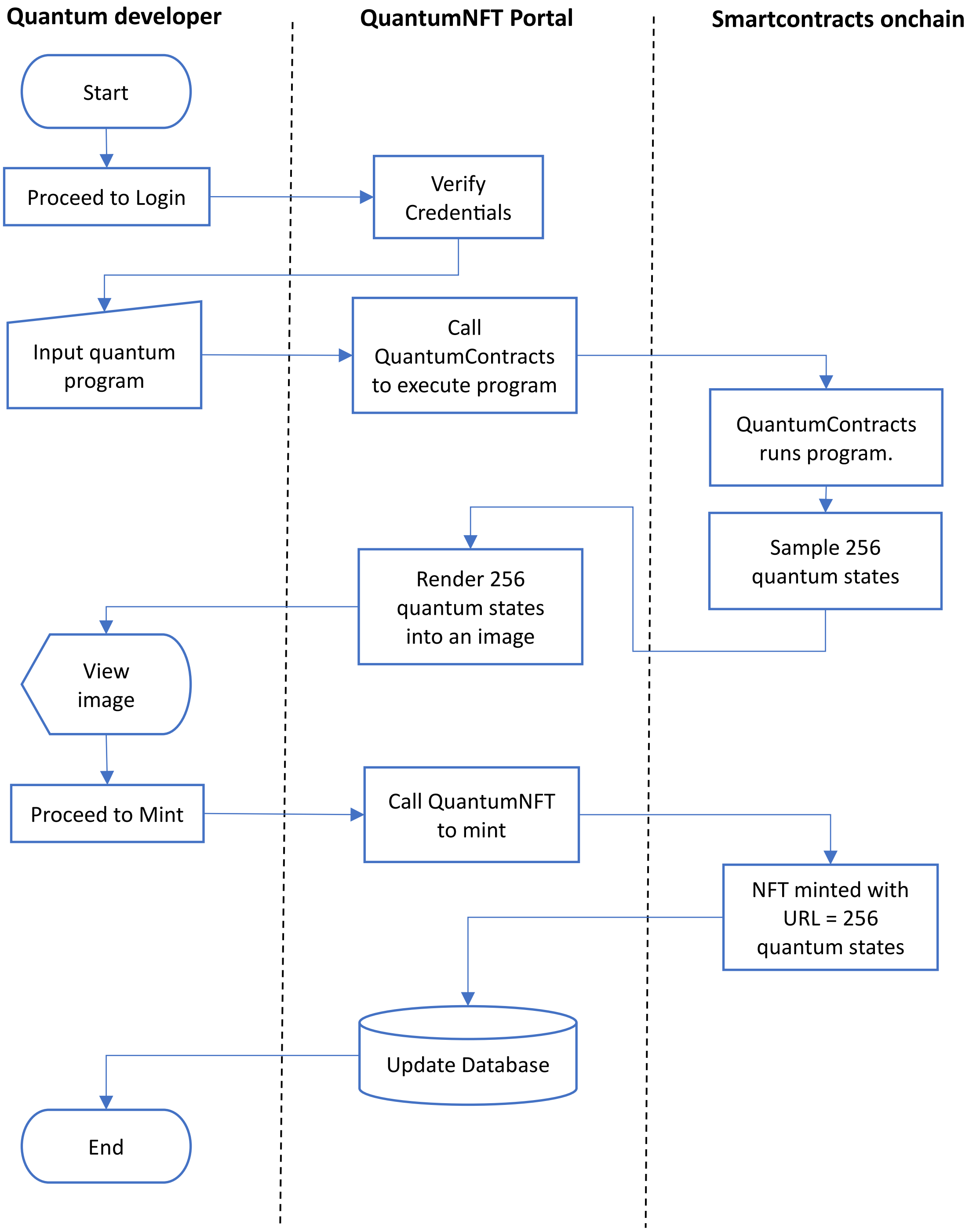

Figure 1 - Walkthrough on how a quantum developer mints a QuantumNFT

1. The quantum developer starts by initiating the login process to establish an identity on the QuantumNFT portal.
2. Once authenticated, the quantum developer submits the quantum program for minting.
3. The portal first calls QuantumContracts, a 8-qubit on-chain quantum emulator smartcontract to execute the quantum program.
4. QuantumContracts will execute the quantum program, and returns the quantum state, which is 256 possible results from the execution. For example, in the case of a 2-qubit bell state, QuantumContracts will return 128 results with “00” values, and 128 results with “11” values.

5. The portal will use the 256-value quantum state to deterministically render an image which is displayed to the quantum developer.
6. The quantum developer can then proceed to execute the mint operate which will store the quantum state on the blockchain. The execution of the quantum program is done separately from the minting of the NFT to reduce the gas fees required since QuantumContracts can be implemented as a view function.
7. The portal will update the database to tag the minted QuantumNFT to the quantum developer.

In the rest of the section, we cover the main implementation aspects of QuantumNFT.

Quantum emulator on-chain

We built "QuantumContracts", a 8-qubit quantum emulator on-chain in order to execute the quantum program submitted by the quantum developer. The language used to write the quantum program is "QuIC" (Quantum Interpreted Circuits) where the BNF notation of QuIC quantum program is as follows.

```
<Program>           ::= <Gate sequence> | <Program><Delim><Gate sequence>
<Gate sequence>     ::= <Gate> | <Gate><Gate sequence>
<Gate>              ::= H | C | N | X | Y | Z | P | p | T | t | I | m
<Delim>             ::= , | .
```

The use of QuIC allows for efficient description and execution of the quantum program which is important since we want to run the quantum program on-chain. The gate operations supported by QuIC are Hadamard (H), Control-Not (CN), Pauli-X (X), Pauli-Y (Y), Pauli-Z (Z), π/2 and π/4 Phase-shift (P, T), Conjugate π/2 and π/4 Phase-shift (p, t), Identity (I) and measurement (m).

We first take the example of a 2-Qubit fully-entangled Bell state. Figure 2 shows the circuit diagram of the quantum program which consists of passing the first qubit through a Hadamard gate, before a Control-Not gate which entangles the first qubit with the second qubit. This creates a fully-entangled state where the possible measurements of the 2 qubits are either "00" or "11". To represent this program in QuIC, we first segment the gates in an ordered sequence before enumerating the gates within the sequence. In Figure 2, sequence 1 consists of a Hadamard gate on qubit 1 and Identity gate on qubit 2, and sequence 2 consists of a Control-Not function spanning from qubit 1 to qubit 2. The eventual QuIC program is thus "`HI,CN.`" where the "," delimiter denotes the next sequence. Some additional examples of QuIC programs of well-known quantum algorithms are:

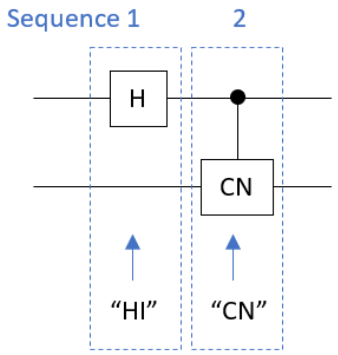

Figure 2 - 2-Qubit Bell-state

- 3-Qubit GHZ :
    - `HII,CNI,ICN.`
- Grover's search for "11" :
    - `HHI,IIX,IIH,XXI,CCN,XXI,IIH,IIX,HHI,XXI,IHI,CNI,IHI,XXI,HHI.`

Token data stored on chain

QuantumNFT stores the following data on-chain:

- TokenID.
    - The TokenID uniquely identifies the NFT that is owned by each quantum developer.
    - It will be made up of <CircuitID> || <Serial Number>.

  - The <CircuitID> uniquely identifies the quantum program using CircuitID = Hash(quantum program). The same quantum program written by different developers will get the same <CircuitID>.
  - The <Serial Number> is an ever increasing number on the total number of NFTs minted. Every NFT minted will cause the <Serial Number> to increment by 1.
- 256-value quantum state.
  - This is the result of 256 executions + measurement of the quantum program that is submitted by the quantum developer, and is a representation of the potential usefulness of the quantum program.
  - The 256 execution results are sorted by qubit values starting with qubit values of all zeros, and ending with qubit values of all ones.

Storing these two values does not reveal the actual quantum program submitted by the quantum developer, but is able to allow a third party to verify the authenticity of the quantum program. A quantum developer can claim knowledge of a particular NFT by providing a quantum program such that:

- Hash(quantum program) = CircuitID which is part of the TokenID
- Executing of quantum program 256 times results in the 256-value quantum state associated with the TokenID

Image representation of quantum state

The visualization of the 256-value quantum state is an important feature within QuantumNFT. Most quantum programming platforms provide graphical representation of the quantum program or quantum circuit in the form of gates similar to figure 2, and tend to depict the results in the form of histogram graphs. We believe there is room for innovation where better representation of the results can lead to better algorithms being written. For the initial version, we make use a 2-dimension 256x256 palette of 24-bit RGB colour to represent the 256-value quantum state. Depending on the number of qubits used in the program, we segment the 24-bit colour for each pixel into equal parts, to be displayed if the qubit value is 1. The segmentation is shown below.

Qubit	Qubit colours when qubit = 1	Remarks
1	White	
2	🟨🟦	12 bits per qubit
3	Red Green Blue	8 bits per qubit
4	🟥🟩🟦⬛	6 bits per qubit
6	🟥⬛🟩⬛🟦⬛	4 bits per qubit
8	🟥⬛🟩🟩⬛🟦🟦⬛	3 bits per qubit

For example, for a 3-qubit program, the 24-bit pixel is equally divided to 8-bits to represent each qubit. If the result has the first qubit = 1 then the first 8 bits of the pixel is turned on, if the second qubit = 1 then the next 8 bits of the pixel is turned on, and similarly if the third qubit = 1 then the final 8 bits of the pixel is turned on. If all three qubits = 0, this means nothing is turned on resulting in a black pixel. If all three qubits = 1, this means all 24 bits are turned on resulting in the presentation of a white pixel. We also divide the image with a number of rows representing the number of qubits in the program. We finally render the image in an animated gif for visual effect. Figure 3 shows the representation a 3-qubit program "HHH." where all 3 qubits are put into a superposition.

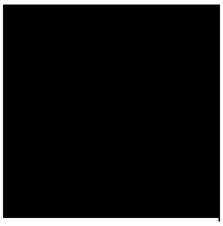

Figure 3 - Quantum state of "HHH"

The advantage of such a representation is its compactness in presenting the results. We give further examples of the image representation of the 3-qubit GHZ algorithm (on the left) and 8 qubits in superposition (on the right) in Figure 5.

3 Qubit GHZ

8 Qubit superposition

Figure 4 - More examples of quantum states

As the QuantumNFT ledger stores the 256-value quantum state, we can, in the future, improve on the images without affecting the NFT that is minted.

Business Model

There are several ways where monetization can happen for QuantumNFT:

- B2C Software-as-a-service.
    - Customer: Quantum developer.
    - We can impose a per-transaction minting fee, in addition to gas fees, whenever a quantum developer mints a QuantumNFT. This is similar to most Web3 smartcontracts offerings.
- B2C Platform-as-a-service.
    - Customer: Quantum developer.
    - We can charge a subscription fee for the use of the QuantumContracts smartcontract to perform quantum-computation on the submitted quantum program. This is similar to most quantum computation cloud providers.
- B2B Community Marketing and advertisement.
    - Customer: Companies in the quantum industry
    - We can charge for advertising banners or other marketing collateral which are displayed on the QuantumNFT portal. This is a similar business model to many marketplaces.
- B2B Data-as-a-service.
    - Customer: Companies wanting to hire quantum developers
    - We can charge for job searches by companies or publicizing job opportunities to the QuantumNFT community. This is a similar business model to many community portals.

At present, we are working on validating the appropriate model for QuantumNFT.

Future Extensions

The planned extensions for QuantumNFT are:

- Supporting other quantum programming languages. At present, QuantumNFT only support QuIC. The QuantumContracts can be extended to support other popular quantum programming languages including IBM Qiskit, CirQ, Microsoft Q#, QML, etc.

- Support more powerful quantum platforms including actual quantum computers. At present, only 8-qubit and below quantum programs are supported. This can be improved to support higher number of qubits and possibly to run the QuantumContracts on actual quantum hardware for performance improvements.
- Solution-bounty. At present, quantum developers are able to write different programs without specific problem statements or objectives to reach. This can be improved to set specific goals or problem to challenge the skill of these developers to reach.