A Deep-Learning-Based Proposal to Aid Users in Quantum Computing Programming

Juan Cruz-Benito[1](✉), Ismael Faro[1], Francisco Martín-Fernández[1], Roberto Therón[2], and Francisco J. García-Peñalvo[2]

[1] IBM Research. T.J. Watson Research Center, Yorktown Heights, NY, USA
{juan.cruz, ismael.farol}@ibm.com, pacom@us.ibm.com

[2] GRIAL Research Group, Department of Computer Science, University of Salamanca, Salamanca, Spain
{theron, fgarcia}@usal.es

Abstract. New languages like Open QASM and SDKs like QISKit open new horizons for the research and development in the new paradigm of quantum computing. Despite that, they present an evident learning curve that could be hard for regular developers and newcomers in the field of quantum computing. On the other hand, currently there are many ways to build intelligent systems that can learn from humans and processes to build a knowledge corpus and provide a different kind of help to humans in tasks like aiding in decision making processes, recommending multimedia resources, building conversational agents, etc. In this paper we describe a work-in-progress project developed by the IBM Q team that implements an intelligent system based on a deep learning approach that learns how people code using the Open QASM language to later offer help and guidance to the coders by recommending different code sequences, logical steps or even small pieces of code. During the paper, we describe our current approach and first results. They include the use of *seq2seq* neural networks that effectively learn quantum-code sequences, and which will be tested in real context in the near future to improve the user experience in IBM Q Experience products.

Keywords: Deep learning · Artificial intelligence · Quantum computing Programming · Open QASM · QISKit

1 Introduction

Quantum computing programming is not currently an easy task. New languages like Open QASM [1] and SDKs like QISKit [2, 3] open new horizons for the research and development in the new paradigm of quantum computing [4, 5]. Despite that, they present a non-easy learning curve for regular developers and newcomers in the field of quantum computing. On the other hand, there are nowadays many ways to build intelligent systems that can learn from humans and build a knowledge corpus on their own. These knowledge corpuses could be used to provide a different kind of help to humans in tasks like aiding in decision making processes, recommending multimedia resources, building conversational agents, etc.

Related to the aforementioned intelligent systems, we find in the literature and new media buzzwords like Artificial Intelligence (AI) [6], Machine Learning (ML) [7, 8], Deep Learning (DL) [9, 10], etc. Many modern applications include these kind of concepts and keywords to look trendy; others involve them to deal with problems that are difficult to solve in other traditional ways. Apart of the trendiness of the terms, it is a fact that these research fields are increasingly present: many enterprises are spending a lot of effort and money to be AI-driven; including AI in applications, decision systems, etc. In this sense and related to the concept of aid provided by AI or intelligent systems, we present our core-concept of User Experience (UX). The UX is commonly defined as "a person's perceptions and responses that result from the use or anticipated use of a product, system or service" [11]. Related to the ISO definition, UX includes all the users' emotions, beliefs, preferences, perceptions, physical and psychological responses, behaviors and accomplishments that occur during, before and after the usage. According to this ISO, there are three factors that influence user experience: system, user and the context of use. Considering these three factors, and related to the new wave related to AI, we can think that it is possible to interfere on the system or the context of use [12] by applying AI to existing systems. Thinking in the AI application, we can help to improve the UX, since the AI could learn from users' and previous usage to change or adapt the system or specific features to the current user's needs, desires and behaviors.

Many times, the UX is merely considered in the context of visual applications (visual UIs) or related to common products designed for regular users. In the case of our work-in-progress research, we try to improve the user experience in the context of programming under the quantum computing paradigm. Learning how people code using the products developed by IBM Research [13], we think is possible to distill knowledge to later use it in guiding and helping other quantum-programmers in common tasks related to the code. In our experience, currently there are many common issues on coding quantum programs (definition of qubits to use, measurement operations, etc.) based on some common rules and patterns, that could represent general (and simple) cases where the users could initially be helped. This help could positively affect the users by fulfilling their desires and expectations to achieve success with their code and experiments or on creating more positive experiences through being helped by an intelligent system that could provide real-time feedback on their code [14, 15].

Gathering all these ideas and core concepts, in this paper we describe a work-in-progress (WIP) project developed by the IBM Q team that implements an intelligent system based on a deep learning approach that learns how people code using the Open QASM language to offer help by recommending different code sequences, logical steps or even small pieces of code. To present this work, section two introduces the core concepts of the *seq2seq* approach and the importance of the natural language processing (NLP) in our context. The third section presents our proposal, describing how our WIP project is developed in terms of technology, details on datasets, etc. as well as our first results achieved. The fourth and fifth sections depict the future work to be done and a brief conclusion on our proposal.

2 *Seq2seq* and the Importance of NLP Approaches

What is source code? In fact, and with no intention of providing a deep definition, the code is a human-readable set of words or alphanumerical characters (instructions) previously defined that could be accompanied by other characters like punctuation, etc., and follows a logical structure and some kind of grammar [16, 17]. Following this consideration (and the *programming languages* idea) we find that the foundations of coding are not so far from those that define the human languages.

In the natural language processing area (NLP) many researchers work using artificial intelligence to analyze human language and design conversational systems, to summarize automatically texts, to learn and replicate communicative styles, etc. In this sense we are using concepts from NLP to teach neural networks how to code using quantum computing languages and libraries (mainly using Open QASM [1]) like the human programmers. That is, we are feeding recurrent neural networks (RNN), as we will explain below, the code entered by programmers when they use the IBM Q Experience [18] to enable them to learn how the code is composed in the context of quantum programming.

Regarding the approach we are following under the NLP umbrella, it is the sequence-to-sequence (*seq2seq*) neural network model [19–21]. This model consists of two RNN's: "one RNN encodes a sequence of symbols into a fixed-length vector representation, and the other decodes the representation into another sequence of symbols. The encoder and decoder of the proposed model are jointly trained to maximize the conditional probability of a target sequence given a source sequence" [20]. That is, using this method is possible to train a system that produces sequences of symbols using an input sequence of symbols (Fig. 1).

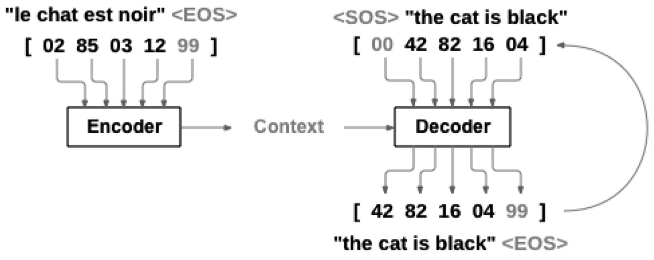


Fig. 1. Overview of a sequence to sequence network using different networks as encoders and decoders. Image taken from [24]

This approach has been used recently with success in the tasks of performing translations between different languages [22, 23] (Fig. 1), since its ability of learn semantically and syntactically meaningful representation of linguistic phrases [20].

In our case, the *seq2seq* neural network is not employed to translate languages, but it is used similarly to translate some sequences to other ones. In our approach, the input sequence will be the one typed by the user in the tools for developing quantum code, and the target sequence will be next logical sentence(s) proposed by the neural net-works. This will be explained further in the following section.

3 Proposal and First Results

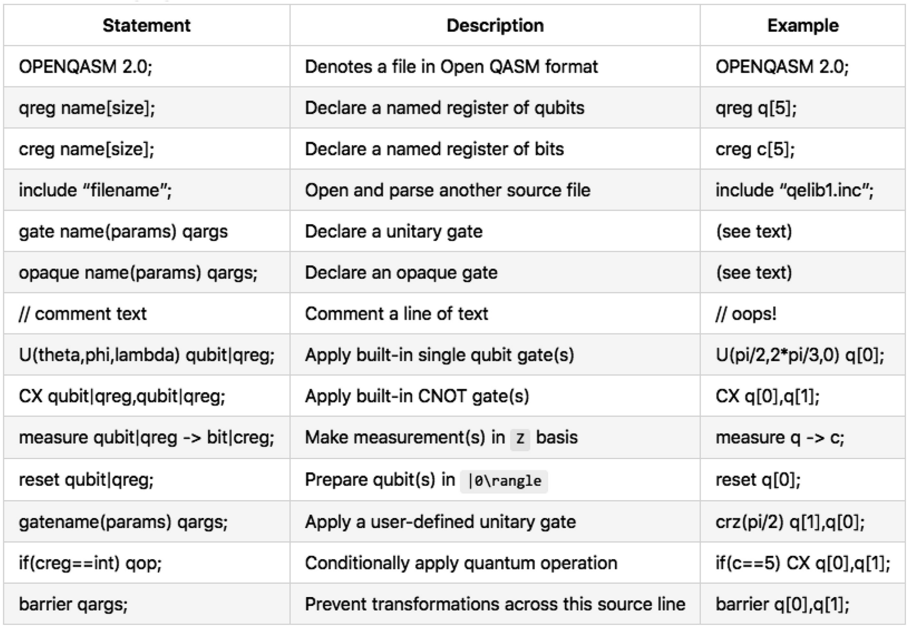
Based on previous experiences by the authors [25] and other researchers [14, 26], the main idea of this paper is that intelligent systems could enhance the user experience. Furthermore, they could enhance the experience of developers while they code new programs in challenging environments like ours. As previously stated, our goal is to provide real-time feedback to IBM Q users by proposing code to them in the different quantum programming environments developed by IBM Research and IBM Q Experience team [18]. In order to provide the recommendations to the users, we are developing a neural network based on a *seq2seq* approach that learns the sequences from the code composed by the users to develop quantum programs in both ways: learning what kind of sentences are used (and relevant for quantum computing) and also the logical sequences they follow to build a quantum program.

To pursue this, there are some important factors to keep in mind: how to develop the *seq2seq* network and its different utilities and how to train it using the data available in our platforms.

First of all, regarding to the technological issues, we use PyTorch [27] as our deep learning framework, supported by the *PyTorch-seq2seq* [28] library from IBM, which is used to build our *seq2seq* network. We selected PyTorch due its features related to the creation of dynamic neural networks and its performance on building deep learning models using GPUs. Also, we use the *seq2seq* library for PyTorch developed by our colleagues at IBM to work from a tested approach in terms of the *seq2seq* models and to avoid starting from scratch for our WIP. This library encloses different functionalities related to the RNN that encodes the input sequences and the decoder RNN that produces the output (target) sequences. Apart of the functionalities it has, we have developed a variation (available in https://github.com/IBM/pytorch-seq2seq/pull/116) on the prediction method used to produce the target sequences. This variation on the prediction method not only produces a unique prediction based on the most probable output sequence (as in the original implementation) but also includes a beam search strategy [19, 29] to produce several possible outputs (target sequences) with variations given an input sequence. In fact, with this change we are replacing the default RNN decoder to use a TopKDecoder RNN.

Secondly, we developed datasets to train our models in the task of predicting the proper code sequence that would be result in adding the next logical sentence to the given code sequence. We built our datasets using the following rule $n -> n + 1$: given a code sequence ‘n’, the predicted sequence would be ‘n + 1’, where ‘n’ is the set of instructions that compose the input sequence, and the ‘+1’ is the following instruction that could be used after the last instruction in the set ‘n’.

The dataset development consisted of two main phases: (1) designing the code sequences to represent inputs and outputs (given and target sequences) and (2) building simpler representations of the code sequences to facilitate the training. On the designing of the code sequences, we augmented the original dataset of quantum programs sent to the IBM Q backends by dividing the quantum program into all the possible parts that followed the rule *n* − > *n* + *1* (input –> output sequence). To reduce the complexity of the training phase, we replaced each unique instruction using a unique key, thereby mapping the instructions used in the code developed by the users to simpler representations. This unique key (composed by 1–3 alphanumerical characters typically), will be used in the datasets to train the encoder and decoder RNN. To build the mapping, we parsed all the defined Open QASM statements and their possible arguments. The list of these Open QASM statements is available in the Fig. 2.



Statement	Description	Example
OPENQASM 2.0;	Denotes a file in Open QASM format	OPENQASM 2.0;
qreg name[size];	Declare a named register of qubits	qreg q[5];
creg name[size];	Declare a named register of bits	creg c[5];
include “filename”;	Open and parse another source file	include “qelib1.inc”;
gate name(params) qargs	Declare a unitary gate	(see text)
opaque name(params) qargs;	Declare an opaque gate	(see text)
// comment text	Comment a line of text	// oops!
U(theta,phi,lambda) qubit|qreg;	Apply built-in single qubit gate(s)	U(pi/2,2*pi/3,0) q[0];
CX qubit|qreg,qubit|qreg;	Apply built-in CNOT gate(s)	CX q[0],q[1];
measure qubit|qreg -> bit|creg;	Make measurement(s) in `z` basis	measure q -> c;
reset qubit|qreg;	Prepare qubit(s) in `|0\rangle`	reset q[0];
gatename(params) qargs;	Apply a user-defined unitary gate	crz(pi/2) q[1],q[0];
if(creg==int) qop;	Conditionally apply quantum operation	if(c==5) CX q[0],q[1];
barrier qargs;	Prevent transformations across this source line	barrier q[0],q[1];

Fig. 2. Open QASM language statements (version 2.0). Figure from [1, 30]

To provide an example of how we are building the dataset main (later separated into train/test/dev datasets) using our *n* − > *n* + *1* rule and the statements mapping, we will use an Open QASM implementation of *Deutsch–Jozsa algorithm* [31] using two qubits (as it appears in [30]).

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[5];
creg c[5];
x q[4];
h q[3];
h q[4];
cx q[3],q[4];
h q[3];
measure q[3] -> c[3];
```

After mapping the code to the simpler keys, we would get the following code (we have removed the *line feed* between the different lines, making the code a single line):

```
O; I; Q5; C5; X4; H3; H4; CX34; H3; M33;
```

Later, we augment for the dataset the mapped code in different lines, where each line follows the *n − > n + 1* rule, and a line below another represents *n = n −1*. So, using the mapped code, we will get the following logical code sequences (Table 1).

Table 1. All the possible logical code sequences for the mapped *Deutsch–Jozsa algorithm* following our augmentation rules

Input sequence (given one)	Output sequence (target)
O; I; Q5; C5; X4; H3; H4; CX34; H3;	O; I; Q5; C5; X4; H3; H4; CX34; H3; M33;
O; I; Q5; C5; X4; H3; H4; CX34;	O; I; Q5; C5; X4; H3; H4; CX34; H3;
O; I; Q5; C5; X4; H3; H4;	O; I; Q5; C5; X4; H3; H4; CX34;
O; I; Q5; C5; X4; H3;	O; I; Q5; C5; X4; H3; H4;
O; I; Q5; C5; X4;	O; I; Q5; C5; X4; H3;
O; I; Q5; C5;	O; I; Q5; C5; X4;
O; I; Q5;	O; I; Q5; C5;
O; I;	O; I; Q5;
O;	O; I;

Using this technique, we think we will be able to teach the *seq2seq* network what are the logical steps in terms of Open QASM statements (and their arguments) used to code a quantum program, in a similar mode to the *seq2seq* training to translate between different languages.

To test this approach, we built some small datasets (typically including several thousands of code sequences) to train our *seq2seq* networks and validate our main idea. In the case of these first tests, we used a mapping algorithm that discards the arguments that accompanied the statements in the original code. This could be useful to validate if the *seq2seq* networks are learning the code grammar properly (which could be easier to discover considering the small size of the initial datasets tested). For example, in a typical quantum program, after defining a register of qubits we defined a register of classical bits. In our tests, we do not mind if the number of qubits is the same as the number of bits or other details. We just want to validate if the neural network has learned that a declaration of a register of classical bits should follow a declaration of a register of qubits. The initial results with the training, using the simpler mapping, show that our *seq2seq* network achieves a prediction accuracy of 88–92% by comparing the training and test datasets (using the default decoder predictor of just one statement more in the target sequence per each sequence input). In our case, we also introduced a quantitative testing of the results provided by the dataset. In this qualitative testing we tried to verify if the results provided by the default predictor and the predictor based on beam search are comprehensive and logical in the context of quantum programming. This qualitative testing is currently ongoing, and it involves some IBM Q team members with expertise in quantum coding who will provide their impressions on the results achieved.

Also, as part of our initial results, we have designed the two main approaches to deploy our intelligent system in real contexts to integrate the assistant in the IBM Q products. These two approaches are the following:

1. Build an API to serve the results of the predictions and submit new codes to continue training our *seq2seq* network. In this case, we follow a *deep learning as a service* approach. The neural network and the different features are available in the cloud through using a REST API (currently implemented using Flask in our case). Using this solution, the different products that will include the intelligent code recommender solution only need to submit the contents of the quantum program being written by the user to obtain what would be the next statement to use.
2. Embed the trained model within the products itself. Considering that the training model is saved into a*.pt* file of about 3 MB, we can embed the file within a website or a program and use it by employing ONNX [32] or introducing our code related to the predictions.

As stated, of these two approaches, we have built only the deployment with the REST API. It will be used in the full validation of the system. In the other case of the embedded code, implementing it will depend on the final success of our full tests.

4 Next Steps and Future Work

The next steps that we will take will be the following:

1. Finalize the internal validation of the simpler version of the Open QASM recommender. This will help us to understand whether the predictions fulfill our expectations or if we need to redefine our intelligent system (tuning the seq2seq network, changing the neural network used, etc.).
2. Build a full dataset to train the neural network. IBM Q has over 2 million records of code executions. We plan to build an augmented dataset using the $n -> n + 1$ rule and include all the code introduced by users to train our future neural networks.

3. Validate internally at IBM Q the final results of these full trainings. At this phase, we will validate the results raised by the neural networks using the simple mapping as well as using a complex mapping (which maps all the statements and their arguments).
4. If all those validations and trainings are successful, we plan to deploy our code recommender using the API REST to be used by the other products and explore the integration of our code into other IBM Q products.
5. Using the deployed version in real products, we will measure different metrics of real user experiences to assess the effect of the intelligent system. In this case, we will measure (prospectively) the ratio between the code offered by the system and those statements employed finally by the users, the time they use to complete their programs using the helper, and the users' opinion through questionnaires, focus groups or other assessment tools typically used in the Human-Computer Interaction research area.
6. Finally, the intelligent system should be re-trained regularly to adapt its knowledge to the new code produced by users. In a novel context like quantum programming, it is possible that the number of users coding as well as the complexity of the code produced will grow. The system will need to be ready to fulfill the expectations of different users with different level of coding skills in quantum computing.

5 Conclusions

This paper describes a work-in-progress project that implements an intelligent system based on a deep learning approach to learn how people code in Open QASM language. This knowledge acquired by the neural networks will be used to offer help and guidance to the programmers by recommending different code sequences, logical steps or even small pieces of code. We intend that this help and guidance during the programming will improve the user experience (UX) of quantum-programmers within the IBM Q products. During the paper we have described our ideas, our current implementation and the different initial results achieved. Among these initial results, we highlight that we have developed *seq2seq* neural networks that are learning quantum code sequences following custom datasets built on our own specification designed to predict possible code sequences given the code produced by users previously. We also provide real examples of how the datasets will be produced and the accuracy achieved by our system in simple test cases. Finally, we described our next steps and the future work that we will try to tackle in the near future.

Acknowledgement. We thank to the ACQX (AI Challenges & Quantum Experience Team at IBM Research) and to the GRIAL research group team for the useful discussions and the feedback received.

References

1. Cross, A.W., Bishop, L.S., Smolin, J.A., Gambetta, J.M.: Open quantum assembly language (2017). arXiv preprint arXiv:1707.03429
2. Cross, A.: The IBM Q experience and QISKit open-source quantum computing software. Bull. Am. Phys. Soc. (2018)
3. IBM Research, QISKit. Quantum Information Software Kit. Accessed 18 Feb 2018. https://www.qiskit.org/
4. Kandala, A., Mezzacapo, A., Temme, K., Takita, M., Brink, M., Chow, J.M., Gambetta, J. M.: Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. Nature **549**, 242 (2017)
5. Ristè, D., da Silva, M.P., Ryan, C.A., Cross, A.W., Córcoles, A.D., Smolin, J.A., Gambetta, J.M., Chow, J.M., Johnson, B.R.: Demonstration of quantum advantage in machine learning. npj Quantum. Information **3**, 16 (2017)
6. Nilsson, N.J.: Principles of artificial intelligence. Morgan Kaufmann, Massachusetts (2014)
7. Raschka, S.: Python machine learning. Packt Publishing Ltd., Birmingham (2015)
8. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V.: Scikit-learn: Machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
9. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning. MIT Press, Cambridge (2016)
10. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**, 436–444 (2015)
11. ISO/DIS: 9241-210: 2010. Ergonomics of human system interaction-Part 210: human-centred design for interactive systems. International Standardization Organization (ISO). Switzerland (2009)
12. ISO/DIS: Draft BS ENISO 9241-220 Ergonomics of human-computer interaction. part 220: processes for enabling, executing and assessing human-centred design within organizations (2016)
13. Castelvecchi, D.: Quantum cloud goes commercial. Nature **543**, 159 (2017)
14. Lee, S., Choi, J.: Enhancing user experience with conversational agent for movie recommendation: Effects of self-disclosure and reciprocity. Int. J. Hum. Comput. Stud. **103**, 95–105 (2017)
15. Shneiderman, S.B., Plaisant, C.: Designing the user interface, 4th edn. Pearson Addison Wesley, USA (2005)
16. The Linux Information Project, Source Code Definition. Accessed 20 Feb 2018. http://www.linfo.org/source_code.html
17. Harman, M.: Why source code analysis and manipulation will always be important. In: 10th IEEE Working Conference on Source Code Analysis and Manipulation (SCAM), pp. 7–19. IEEE (2010)
18. IBM Research, The IBM Quantum Experience. Accessed 18 Feb 2018. https://www.research.ibm.com/ibm-q/
19. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Advances in neural information processing systems, pp. 3104–3112 (2014)
20. Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder-decoder for statistical machine translation (2014). arXiv preprint. arXiv:1406.1078
21. Vinyals, O., Le, Q.: A neural conversational model (2015). arXiv preprint. arXiv:1506.05869 (2015)
22. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate (2014). arXiv preprint. arXiv:1409.0473

23. Neubig, G.: Neural machine translation and sequence-to-sequence models: A tutorial (2017). arXiv preprint. arXiv:1703.01619
24. Robertson, S.: Translation with a Sequence to Sequence Network and Attention. Accessed 20 Feb 2018. http://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html
25. Cruz-Benito, J., Vázquez-Ingelmo, A., Sánchez-Prieto, J.C., Therón, R., García-Peñalvo, F. J., Martín-González, M.: Enabling adaptability in web forms based on user characteristics detection through A/B testing and machine learning. IEEE Access. **6**, 2251–2265 (2018)
26. Loup-Escande, E., Frenoy, R., Poplimont, G., Thouvenin, I., Gapenne, O., Megalakaki, O.: Contributions of mixed reality in a calligraphy learning task: Effects of supplementary visual feedback and expertise on cognitive load, user experience and gestural performance. Comput. Hum. Behav. **75**, 42–49 (2017)
27. Paszke, A., Gross, S., Chintala, S., Chanan, G.: PyTorch: Tensors and Dynamic neural networks in Python with strong GPU acceleration. Accessed 20 Feb 2018. https://github.com/pytorch/pytorch, IBM Research, pytorch-seq2seq. Accessed 20 Feb 2018. https://github.com/IBM/pytorch-seq2seq
28. Wiseman, S., Rush, A.M.: Sequence-to-sequence learning as beam-search optimization (2016). arXiv preprint arXiv:1606.02960
29. IBM Research, Open QASM. Gate and operation specification for quantum circuits. Accessed 20 Feb 2018. https://github.com/QISKit/openqasm
30. Deutsch, D., Jozsa, R.: Rapid solution of problems by quantum computation. In: Proceedings of the Royal Society of London. A, pp. 553–558. The Royal Society (1992)
31. Open Neural Network Exchange, ONNX Github repository. Accessed 20 feb 2018. https://github.com/onnx/onnx