Classical and Quantum RNN for Java Code Comprehension Task

Mobeen Alhalabi[1], and Fathy Eassa [2]

[1]Department of Computer Science, Faculty of Computing and Information Technology, King Abdulaziz University (KAU), Jeddah 21589, Saudi Arabia (e-mail: malhalabi0005@stu.kau.edu.sa)
[2]Department of Computer Science, Faculty of Computing and Information Technology, King Abdulaziz University (KAU), Jeddah 21589, Saudi Arabia (e-mail: Feassa@kau.edu.sa)

Abstract

Program comprehension refers to the process of extracting relevant information from programs through analysis, abstraction, and reasoning. It is a crucial aspect of software development, maintenance, migration, and other related processes. Traditionally, program comprehension has relied heavily on developers' experience. However, as software systems become increasingly large and complex, it is time-consuming and challenging to depend solely on a developer's prior knowledge to identify program features. Furthermore, fully uncovering the hidden features within the program becomes difficult.

Deep learning offers a data-driven, end-to-end approach that utilizes deep neural networks to analyze existing data and discover these hidden features. We can automatically learn the underlying features implied in programs by applying deep learning techniques to program comprehension. This approach allows us to fully leverage the knowledge embedded in the program, ultimately improving the efficiency of program comprehension.

Among the various areas related to language processing, programming languages stand out as a notable application of modeling techniques. For years, the machine learning community has investigated this field of software engineering, focusing on goals such as auto-completing, generating, fixing, or evaluating code written by humans. With the increasing popularity of deep learning-enabled language models, we identified a lack of empirical studies comparing deep learning architectures for creating and utilizing language models based on programming code.

This paper implements several recurrent neural network architectures, including RNN, LSTM, GRU, and bi-directional variations, along with quantum algorithms applied to each in series: QRNN, QLSTM, QGRU, and QDI. We utilize transfer learning and tokenization to evaluate how well these architectures perform in building language models using a Java dataset for code generation and summarization.

This comparative study discusses the architecture of each model. It presents the results obtained, highlighting gaps discovered in evaluating language models and their application in real programming contexts.

Keywords: Program Comprehension, Recurrent Neural Network, Deep Learning, Java Source Code, Natural Language Processing, Quantum Neural Network, Quantum Physics.

1. Introduction

The high energy physics (HEP) community has a longstanding tradition of managing extensive datasets and employing sophisticated statistical methods to examine experimental data across the energy,

intensity, and cosmic frontiers. The HEP community requires substantial computational advancement to manage escalating data quantities, and quantum information science (QIS) innovations may offer a feasible answer. Quantum supremacy refers to the capability of solving problems more rapidly than any classical approach. In the context of computational complexity theory, this often signifies delivering super-polynomial acceleration compared to the most efficient known classical method. In this study, we will define quantum advantage as the superiority of quantum devices over classical ones, not necessarily characterized by super-polynomial speedup. Machine Learning (ML) techniques provide significant advantages for scalable data analysis. In conjunction with big data and graphics processing unit (GPU) capabilities, DL has markedly enhanced the capacity to evaluate extensive quantities of text data [1].

Nevertheless, quantum computing has not yet achieved substantial advancements in implementing robust representation learning algorithms. However, it is crucial to investigate the potential of quantum ML, which has lately garnered significant attention. This research introduces a novel hybrid four different quantum recurrent neural network (QRNN) framework to illustrate the quantum advantage over equivalent recurrent techniques. We evaluate its efficacy in classifying high-energy physics events using simulated data from neutrino investigations. We demonstrate that for a comparable number of parameters in the QRNN and traditional RNNs, the QRNN can achieve quicker learning or attain superior testing accuracy with fewer training epochs with some of the algorithms; on the other hand, some other types of recurrent methods; the quantum and classical technique were close to each other in the accuracy metric. Our simulations indicate a potential empirical quantum advantage of the four types of QRNNs over RNNs regarding testing accuracy [2].

Our classical knowledge derives from our life experiences; nevertheless, these experiences do not encapsulate the fundamental mechanisms of nature. Quantum mechanics, the fundamental principles governing our reality, account for all observable phenomena. The phenomena of quantum mechanics are incongruent with conventional reasoning. Quantum computing employs the principles of quantum mechanics to perform computational tasks; quantum information pertains to using quantum phenomena to enhance information transfer, storage, and retrieval, including developing sophisticated cryptographic techniques. The many permutations associated with quantum processing enable quantum computers to double their memory capacity by adding each qubit, which is the source of quantum computing's power. Quantum computers can resolve problems unattainable by classical computers, exemplified by their capacity to factor enormous numbers by Shor's algorithm. Moreover, in certain instances, integrating classical and quantum computers might yield enhanced problem-solving efficiency, with quantum algorithms demonstrating superiority over classical counterparts [2].

Quantum ML involves leveraging the potential of quantum computers to develop scalable machine learning models that surpass conventional models' performance while being more cost-effective [3]. The principles of quantum computing (QC) investigate the challenges associated with data storage, processing, and analysis [4]. Quantum mechanical systems are established by transforming information, which is consistently designated as quantum information. Quantum information denotes data that represents the state of a quantum system [5]. The fundamental concept of quantum information pertains to the state of any quantum system characterized by two distinguishable degrees of freedom; the logical values 0 and 1 are referred to as a Qubit. A quantum computer exhibits the counterintuitive phenomena of quantum physics, such as superposition, entanglement, and tunneling. Consequently, a quantum computer can swiftly address issues that exceed the capabilities of classical devices [6].

Quantum computing is a rapidly advancing field that utilizes the principles of quantum mechanics to tackle complex problems that pose challenges for classical computing. Machine learning is a specialized area within artificial intelligence that enables computers to identify patterns based on their experiences. The rapid increase in data volume presents challenges for traditional machine learning algorithms in handling big data, while quantum computing offers the potential for significantly faster processing capabilities. Integrating quantum computing with machine learning has led to the emergence of a novel domain called quantum machine learning. Quantum machine learning algorithms leverage the rapid processing capabilities of quantum computing, demonstrating a significant speed advantage over

classical methods. Natural language processing (NLP) represents a significant domain within artificial intelligence, facilitating the computer's comprehension of human languages. Currently, efforts are underway to leverage the speed advantages of quantum machine learning in applications related to natural language processing [7].
This paper is organized as follows: Section 2 discusses the literature review of previous related work; Section 3 introduces the data used in this paper; Section 4 shows the algorithms' performance on the experimental data used; and Section 5 is focused on results and discussion. Finally, Section 6 discusses our conclusion and future work.

2. Literature Review

The first study that utilized the MNIST dataset in conjunction with the QRNN method achieved an accuracy range of 94.6% to 96.7% [8]. Undoubtedly, current recurrent models—even basic RNNs—surpass the suggested QRNN architecture in practical learning tasks. It was stated that simulating a substantial number of qubits on classical hardware is challenging due to the exponential growth of memory requirements relative to the workspace size, which constrains the number of parameters that can be incorporated into our model. Conversely, this overhead would dissipate on a quantum computer, leading to a linear execution time concerning circuit depth. Furthermore, the authors in [8] assert that the topology of the proposed QRNN cell is influenced by the actions of its distinct phases (input writing, processing, output writing). Although the resultant circuits exhibit significantly greater structure than current VQE configurations, the architecture remains rudimentary compared to the numerous elements of an RNN, not to mention an LSTM. A more specialized circuit architecture will likely surpass the "simple" quantum recurrent network discussed herein.
The work [9] implements the QRNN mechanism, and the efficacy of the current QRNN models is validated using three distinct types of classical sequential data: meteorological indicators, stock prices, and text classification. The models attain accuracies ranging from 98.6% to 99.08%.

[10] used the QRNN algorithm in conjunction with the MNIST dataset, attaining an accuracy of 85%. Homodyne detection has exceptional efficiency and is executed using photodiodes and electronic components. The authors stated that to achieve the necessary nonlinearity for the activation function, they utilized the tensor product structure of the quantum circuit in conjunction with measurement, hence eliminating the need for extra nonlinear components. This platform's unparalleled advantage lies in its genuine quantum operation, eliminating the necessity of measuring reference qubits or qu-modes throughout each algorithm iteration to enforce the internal rules of the hidden layer. Nevertheless, measuring certain qu-modes and utilizing this output for a subsequent iteration is feasible.

On the other hand, [11] employed QRNN and QSANN for experiments involving text data. The QRNN model, among others, was evaluated on the Meaning Classification (MC) and Relative Pronoun (RP) datasets and the Amazon, Yelp, and IMDB datasets. QRNN accuracy was: MC, RP, Amazon, IMDB, Yelp: 0.986, 0.613, 0.658, 0.859, 0.602. QSANN respectively was: 1, 0.93, 0.64, 0.57, 0.72. The study emphasized a segment of the current research on quantum sequential models, such as QRNNs and quantum self-attention models, demonstrating the applicability and versatility of QSANNs in text and image classification tasks. QSANN, augmented with positional encoding, demonstrates enhanced performance through its self-attention mechanism, highlighting critical words or key components within an image or text sequence. This mechanism enables the model to identify and prioritize crucial information efficiently. Conversely, QRNN encounters difficulties in effectively capturing long-term dependencies because of its built-in short-term memory constraints. Classical self-attention demonstrates exceptional performance across various tasks, such as text summarization and machine translation, by providing an adaptive and contextualized representation of the input material. This represents a significant advancement in modern deep learning systems due to its parallelization and adaptability. Quantum self-attention has the potential to enhance and broaden the properties, transforming machine learning applications by potentially facilitating faster and more precise solutions to complex modeling tasks.

[12] implemented LSTM and QLSTM with the IMDB dataset. Given the intricate nature of simulating the quantum circuit, the training process required approximately 10 minutes, in contrast to just 10 seconds for the classical approach. Upon completing the 300 epochs, the classical network exhibits a loss of 0.053, whereas the quantum network shows a loss of 0.056. It is important to observe that the quantum LSTM employs fewer than half the parameters of its purely classical counterpart (199 compared to 477), all while maintaining equivalent overall performance.

In another study, CNN, Bi-LSTM, and QRNN algorithms were applied to the TREC question classification dataset [13]. The Baseline Logistic Regression model showed commendable performance on the TREC 10 dataset; however, the two-layer stacked Bi-LSTM model exhibited slightly superior results, enhancing the baseline outcome by 1%. The CNN-based method utilizing kernel sizes ranging from 2 to 6, along with a single fully connected layer, demonstrates superior performance compared to all other models, achieving an accuracy of 90.7%. The more intricate QRNN model excels on TREC 10, but simpler options underperform on this dataset, falling short of the baseline model. This may be attributable to a limited training dataset for this task.

[14] built the LSTM and GRU algorithms, utilizing a dataset from Facebook research, available in code-mixed Hindi. The dataset has twenty distinct challenges illustrating various methods for extracting solutions from a collection of provided sentences inside a paragraph. The findings demonstrate the applicability of the GRU, which had superior accuracy on the majority of tasks in comparison to LSTM. The computational complexity of GRU is significantly lower than that of LSTM due to the absence of a gate in GRU. This research concludes that GRU networks are optimal for addressing question-answering or machine comprehension challenges in code-mixed Indian languages. The results achieved for both networks surpass those of other existing models for machine understanding in code-mixed Indian languages.

Code-mixed datasets comprise textual or auditory data collections incorporating a blend of two or more languages or linguistic variants inside a single discourse or communication. These datasets are typically employed in NLP tasks, including language identification, machine translation, named entity recognition, text summarization, sentiment analysis, and speech recognition. The fundamentals of quantum computing and the core concept of quantum machine learning were studied in [7], where Parts of Speech (POS) tagging for code-mixed language in social media utilizing classical LSTM and QLSTM methodologies was proposed. It was shown that QLSTM outperforms classical LSTM for Facebook (Hindi–English, Telugu–English), WhatsApp (Telugu–English), and Twitter (Hindi–English, Bengali–English). The average accuracy of the LSTM model ranges from 38.62% to 47.75%, whereas the average accuracy of the QLSTM model ranges from 45.38% to 59.45% [7].

3. Dataset

DeepCom has published a dataset containing 588,108 pairs of Java methods and their corresponding documentation, sourced from 9,714 GitHub projects for code summarization. In this dataset, the initial phrase of the documentation is considered the method's overview, as it typically outlines the functionality of the Java methods. The descriptions that are empty or consist of a single word, along with setter, getter, constructor, and test methods, due to their simplicity in being modeled, were excluded [15]. DeepCom is an attention-based Seq2Seq model designed to generate comments for Java methods. It utilizes Abstract Syntax Trees (ASTs) as input and employs the SBT method to transform these ASTs into sequences in a specific format. To evaluate the performance of the different algorithms with the DeepCom, we re-implemented these models in our study using the DeepCOM dataset, adhering to the identical parameter settings established in the corresponding research [16]. DeepCom utilizes Natural Language Processing (NLP) methodologies to extract knowledge from an extensive code corpus and produces comments based on the acquired characteristics [17].

4. Methodology

We proposed eight models to evaluate the code comprehension task using the same dataset. These models include classical and quantum versions of RNN, LSTM, GRU, and Bi-Directional RNN. Each model had a series of classification accuracy metrics outlined below. The proposed strategy is organized into 10 phases:

- Data Preprocessing
- Tokenizer
- RNN algorithm
- QRNN algorithm
- Classical RNN-LSTM algorithm (CLSTM)
- QRNN-LSTM algorithm
- Classical RNN-LSTM-GRU algorithm (CLSTM-GRU)
- QRNN-LSTM-GRU algorithm
- Classical BI Directional RNN algorithm (CBI-RNN)
- QRNN BI Directional algorithm (QBI-RNN)

For all eight different models, data preprocessing and tokenization occur before each model begins the evaluation process. Therefore, these two processes are crucial for every model. Below is an explanation of how these processes are carried out, followed by a comparative study of the various models.

4.1 Data Preprocessing

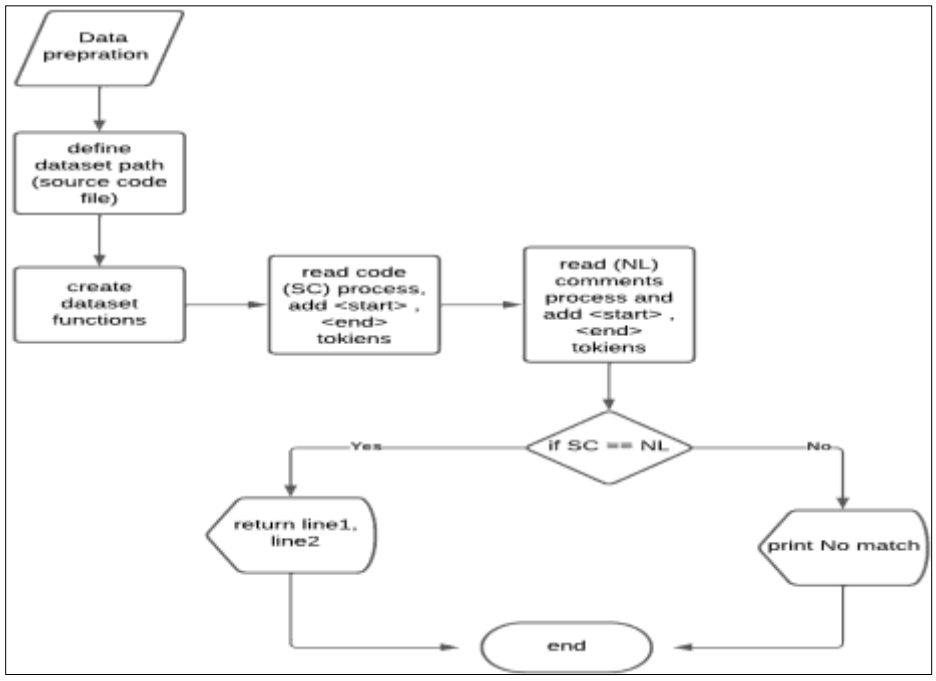
The code defines a function called 'create_dataset' that takes two file paths as arguments: one for the code and another for natural language (NL) comments. The function examines these files and prepares them for further use. After that, the data preprocessing phase accepts the file path within the function, which includes variables that store the paths to the input files. One file contains the code, while the other contains the corresponding NL comments.

The subsequent process in that data preprocessing phase involves defining the function create_dataset, which accepts two parameters: a: the path to the code file, and b: the path to the natural language file. Perusal The Code File follows in this step, which opens the supplied code file and reads its lines into a list named lines1, with each line in lines1 being handled as a: The newline character (\n) is removed after each line. Each line is enclosed with tokens, signifying the commencement and conclusion of each code snippet.

Subsequent reading of the Natural Language File is as follows: Analogous to the preceding block, this code accesses the NL comments file designated by b and imports its lines into a list named lines2. Each line undergoes identical processing, encapsulated with tokens. Subsequently, the data length is verified. After processing both files, the program verifies whether the quantity of code snippets (lines in lines1) corresponds to the amount of natural language comments (lines2). If the counts are incongruent, a warning message is generated. A Return Statement is established to return two lists: i) lines1, which contain the processed code snippets, and ii) lines2, which contain the analyzed natural language remarks. The 'create_dataset' utility efficiently prepares a dataset by reading code and natural language comment files.

Enclosing each line with tokens enhances processing in subsequent tasks, such as building a sequence-to-sequence model. Verify the consistency of code snippets and comments before returning the processed lists. Our machine-learning tasks generally require this in terms of code summary and comment generation. This process is detailed in Figure 1, the flowchart diagram.

Figure 1: Data preprocessing process.



4.2 Tokenizer

This work uses TensorFlow's tf.keras.preprocessing.text; it is a function that tokenizes text data by processing the text, constraining the vocabulary, generating a mapping table, and substituting unknown words. Initial segment Establish Constants: code maxlen = 150; Maximum length for code snippets (150 tokens); nl_maxlen = 30. The maximum length for natural language comments is 30 tokens, with a vocabulary limit of 30,000 words. After that, the function for tokenization is defined as def tokenize (lang, maxlen), where lang represents the textual material designated for tokenization (which may include code or natural language), and maxlen is the upper limit for the sequence lengths post-tokenization. The subsequent step in the process involves the creation of a tokenizer using tf.keras.preprocessing.text.

Tokenizer by instantiating a tokenizer with the following parameters:

- num_words set to vocab maxlen +1. It restricts the vocabulary size to 30,001 words (including the OOV (out of vocabulary) token),
- filters set to filters="". This implies no character filtering (normal behavior would exclude punctuation and special characters);
- oov_token is set to" unk", designating a token for out-of-vocabulary words (words absent from the vocabulary).

After that, the tokenizer is applied to the text data. This process examines the input text (language) and builds a vocabulary based on the frequency of each word. Each unique word is assigned a specific index. The input text is then converted into sequences of integers, where each integer corresponds to a word's index in the vocabulary. For example, "hello" might be represented by the number five if it is the fifth entry in the vocabulary.

Furthermore, the Pad and Truncate Sequences are applied as follows:

- pad sequences guarantee uniformity in sequence length;
- maxlen enforces the maximum length defined in the function call;

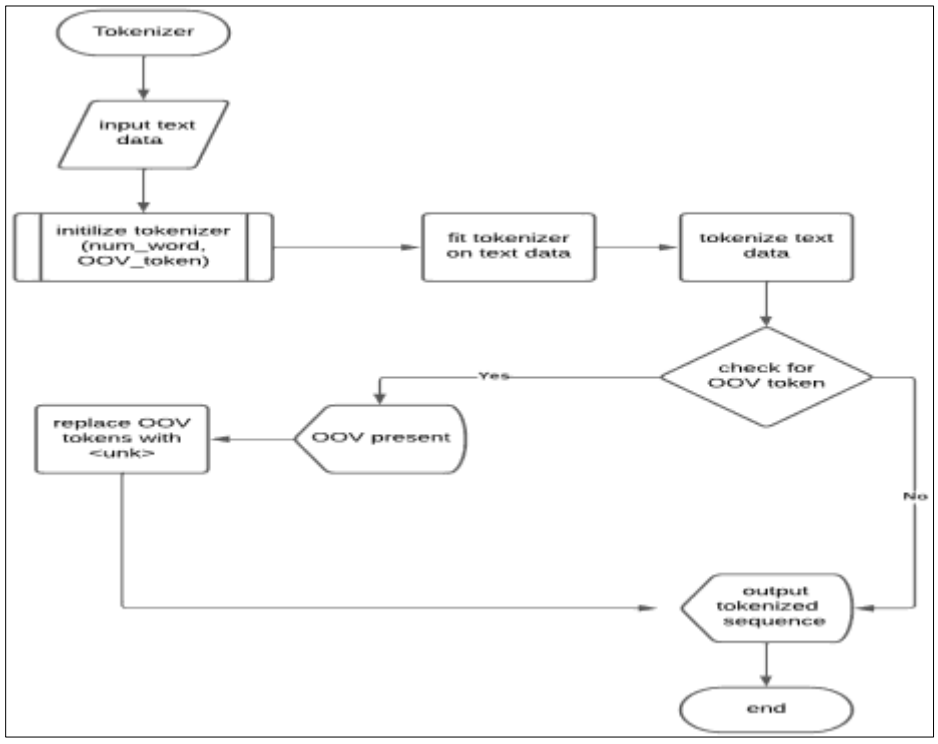
- padding is set to' post'. This appends padding to the end of the sequences;
- Truncating is set to' post'. This process removes any excess sequences from the end when they exceed the maximum length. The final function is the Return Statement, which produces a tensor.
  The tokenized sequences are padded and truncated using `tf.keras.preprocessing.sequence.pad_sequences`. This ensures that all sequences have a uniform length, with `code_maxlen` for code sequences and `nl_maxlen` for natural language sequences.

Padding is added at the end (post padding), and if a sequence exceeds the defined length, it is truncated. The tokenize function is intended to preprocess textual material for application in ML models as follows:

- Initializes a tokenizer that constrains vocabulary and specifies an out-of-vocabulary token.
- Trains the tokenizer on the supplied text data to establish a vocabulary.
- Transforms the text into integer sequences according to the vocabulary.
- Pads or truncates these sequences to achieve uniform length.

This procedure is crucial for formatting text data for deep learning models, which necessitate fixed-size input.
Figure 2 graphically represents this process as a flowchart diagram.

Figure 2: Tokenizer process



5. Results and discussion

Based on **Table 1** performance metrics for various code comprehension models**,** we can perform a comparative analysis to evaluate their effectiveness. The metrics include training accuracy**,** testing accuracy**,** training loss, and testing loss. This section discusses implications, trade-offs, and practical recommendations for this comparative study for each model. Moreover, the statistical significance of the differences in performance and the potential reasons behind the observed trends are addressed. Also, we will briefly compare the strengths and weaknesses of each model, focusing on the importance of classical

and quantum in this study.

Table 1: Performance metrics

Model	Training accuracy	Testing accuracy	Training Loss	Testing Loss
CRNN	0.7760	0.7818	2.8508	3.0025
QRNN	0.7938	0.7948	2.6635	2.8088
Classical LSTM	0.7790	0.7867	3.2855	2.9625
QLSTM	0.7798	0.7886	2.3089	2.3223
Classical GRU	0.7741	0.7803	2.8513	3.3042
QGRU	0.7859	0.7912	2.7164	3.0783
Classical BI Directional RNN	0.9918	0.9951	2.0287	2.0495
QRNN BI Directional	0.9932	0.9950	2.0241	2.1224

Quantum models (QRNN) consistently outperform their classical counterparts in accuracy and loss metrics. Their architectures may offer feature extraction and pattern recognition advantages, potentially due to their ability to more effectively model complex relationships in the data. The differences in accuracy and loss are slight but consistent across all models. This suggests that the quantum advantage, while present, may not be dramatic for this specific task and dataset. QRNN-based models would be the best to use when computational resources are available due to their consistent performance improvements. QRNN-based models consistently achieve higher accuracy and lower training and testing loss, indicating better optimization. Therefore, quantum models may offer better feature extraction and generalization because they can model complex relationships. However, quantum models may require more computational resources for training and inference, and implementing and tuning quantum models can be more challenging than classical models.

Therefore, QRNN-based models should be used if computational resources are available and the task requires high accuracy; otherwise, classical models (especially bidirectional RNNs) may still be sufficient for simpler tasks or resource-constrained environments.

Bidirectional RNNs (classical and quantum) achieve the highest accuracy and lowest loss. It captures context from past and future tokens**,** which is beneficial for code comprehension tasks where understanding the full context of a code snippet is critical. The performance gap between bidirectional and unidirectional models is significant (e.g., testing accuracy of 0.9951 vs. ~0.78-0.79 for unidirectional models). This indicates that bidirectional models are far superior for this task and that using bidirectional RNNs (preferably QRNN-based) for code comprehension tasks where context is crucial.

On the other hand, models with LSTM or GRU layers generally perform better than basic RNNs. They are better at capturing long-range dependencies in the code, which is essential for understanding complex code structures. The improvements are moderate but consistent. For example, QRNN LSTM achieves a testing accuracy of 0.7886 compared to 0.7948 for QRNN without LSTM. In short, using LSTM or GRU layers improves performance, especially for tasks involving extended code snippets or complex logic.

Some models (e.g., QRNN LSTM GRU) show a slight increase in testing loss compared to training loss, indicating potential overfitting. Overfitting occurs when the model learns noise or specific patterns in the training data that do not generalize well to unseen data. The difference between training and testing loss is small but noticeable (e.g., 2.7164 vs. 3.0783 for QRNN-LSTM-GRU). This suggests mild overfitting. The bidirectional models (CRNN and QRNN) achieve remarkable accuracy levels above 99%, indicating they can effectively learn and generalize from the training data. QRNN models outperform CRNN models, demonstrating an advantage in capturing sequential dependencies. Lower loss values in QRNN models suggest better generalization capabilities.

For instance, QRNN-LSTM has significantly lower training loss than its classical counterpart, implying it learns more efficiently from the data. The training loss for the CLSTM-GRU is notably high, suggesting it may be overfitting or struggling with the complexity of the dataset. According to that, if high accuracy in code comprehension is paramount, bidirectional models are the best choice. They leverage context from both directions, which is vital for understanding relationships in code. On the other hand, QRNNs provide a good balance between complexity and performance, making them suitable for scenarios where training time and resource consumption are concerns. Moreover, as codebases grow, QRNNs and bidirectional models may scale better due to their superior understanding of sequential relationships.

5.1 Evaluation and Analysis

5.1.1 Analysis of Classical RNN and Quantum RNN

Figures 3 and 4 present the classical and quantum RNN loss and accuracy. It can be observed that the QRNN consistently outperformed the CRNN across all key metrics. QRNN achieved higher training accuracy (~0.795) than CRNN (~0.7825), indicating better learning capabilities. Similarly, QRNN demonstrated superior validation accuracy (~0.790) compared to CRNN (~0.7800), suggesting better generalization to unseen data. In terms of loss, QRNN also performed better, with lower training loss (~2.6635) and validation loss (~2.8088) compared to CRNN (~2.8508 and ~3.0025, respectively). These results suggest that the quantum-inspired principles employed by QRNN enable more effective feature extraction and pattern recognition, making it better suited for complex code comprehension tasks.

Figure 3: Classical RNN Model Loss and Accuracy

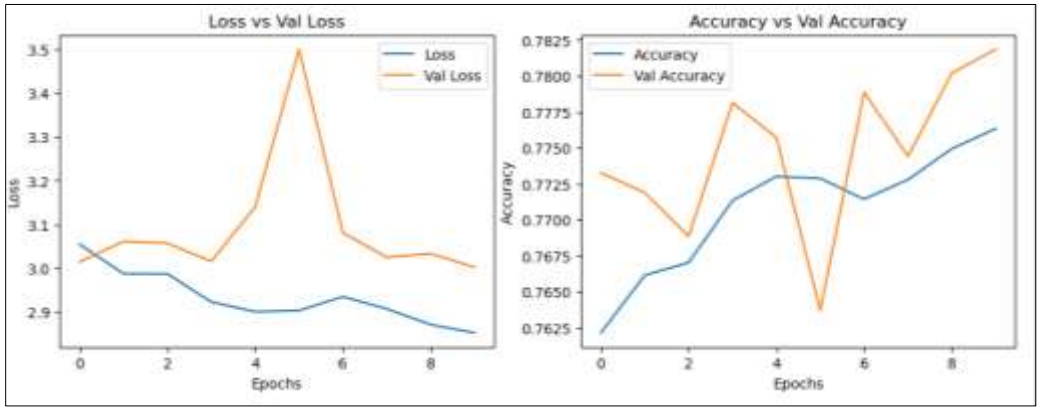
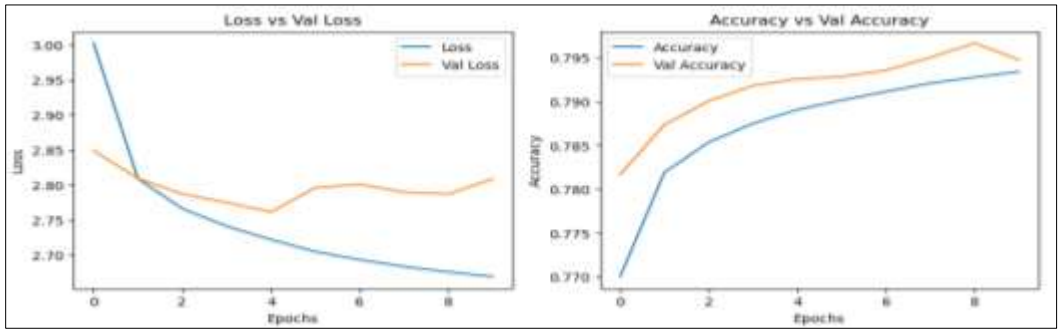


Figure 4: QRNN Model Loss and Accuracy



However, both models exhibited overfitting, as evidenced by the plateauing or slight increase in validation loss and the stabilization of validation accuracy after a certain number of epochs. This behavior was more

pronounced in the CRNN, which struggled to generalize beyond the training data. While QRNN also showed mild overfitting, its impact was less severe, likely due to the model's ability to capture more complex patterns in the data.

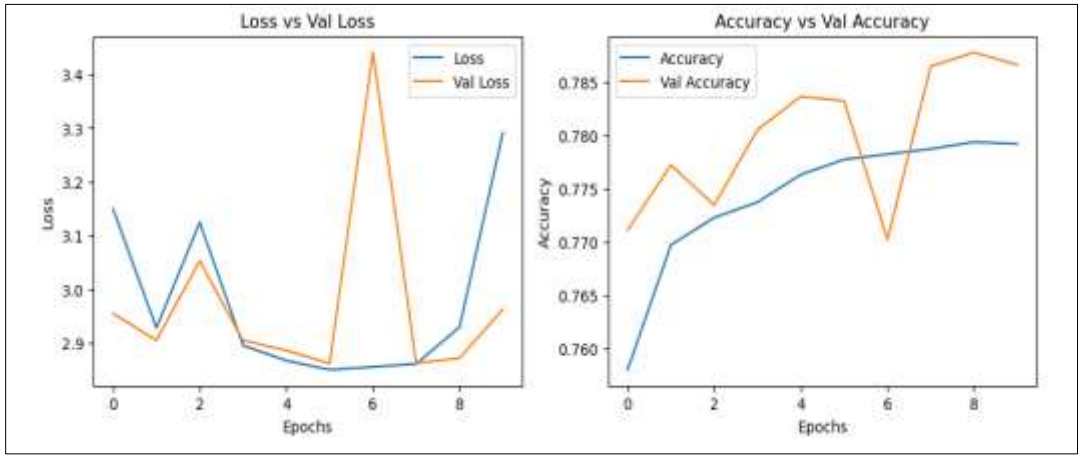
In terms of computational cost, the CRNN has a clear advantage. Implementing it is simpler and requires fewer computational resources, making it suitable for resource-constrained environments or simpler tasks. On the other hand, QRNN is more computationally expensive due to its quantum-inspired architecture, which may limit its practicality for large-scale applications without access to specialized hardware. Despite this, QRNN's superior performance makes it a compelling choice for tasks that demand high accuracy and robust generalization.

Therefore, the QRNN performs better than the CRNN for code comprehension tasks, achieving higher accuracy and lower loss. However, its higher computational cost and complexity must be carefully considered. The CRNN remains a viable option for more straightforward tasks or environments with limited resources. By addressing their respective weaknesses—such as overfitting and computational efficiency—both models can be further refined to meet the demands of specific use cases.

5.1.2 Analysis of Classical LSTM and Quantum LSTM

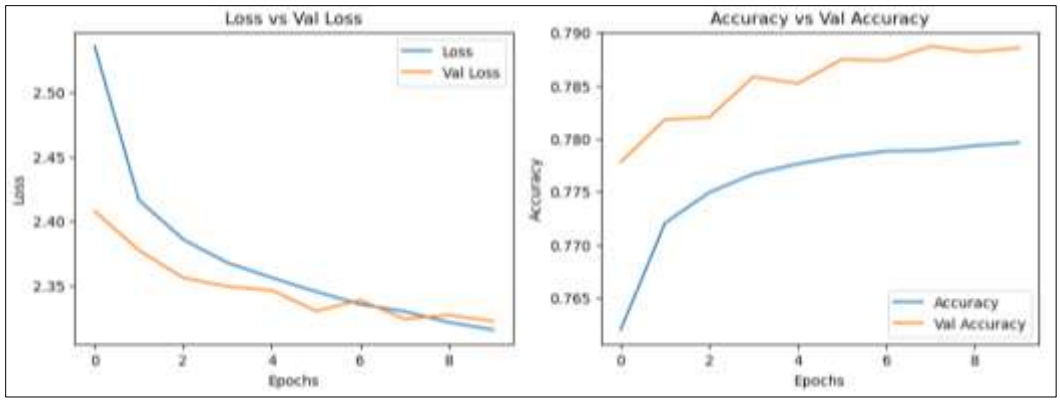
Figures 5 and 6 present the Classical and Quantum LSTM model loss and accuracy. As it can be observed, the QLSTM consistently outperformed the CLSTM across all key metrics. QLSTM achieved higher training accuracy (~0.790) than CLSTM (~0.785), indicating better learning capabilities. Similarly, QLSTM demonstrated superior validation accuracy (~0.785) compared to CLSTM (~0.780), suggesting better generalization to unseen data. In terms of loss, QLSTM also performed better, with lower training loss (~2.0241) and validation loss (~2.1224) compared to CLSTM (~2.3089 and ~2.3223, respectively). These results suggest that the quantum-inspired principles employed by QLSTM enable more effective feature extraction and pattern recognition, making it better suited for complex code comprehension tasks.

Figure 5: Classical RNN LSTM Model Loss and Accuracy



However, both models exhibited overfitting, as evidenced by the plateauing or slight increase in validation loss and the stabilization of validation accuracy after a certain number of epochs. This behavior was more pronounced in the CLSTM**,** which struggled to generalize beyond the training data. While QLSTM also showed mild overfitting, its impact was less severe, likely due to the model's ability to capture more complex patterns in the data.

Figure 6: QRNN LSTM Model Loss and Accuracy



In terms of computational cost, the CLSTM has a clear advantage. Implementing it is simpler and requires fewer computational resources, making it suitable for resource-constrained environments or simpler tasks. On the other hand, QLSTM is more computationally expensive due to its quantum-inspired architecture, which may limit its practicality for large-scale applications without access to specialized hardware. Despite this, QLSTM's superior performance makes it a compelling choice for tasks that demand high accuracy and robust generalization.

The QLSTM performs better than the CLSTM for code comprehension tasks, achieving higher accuracy and lower loss. However, its higher computational cost and complexity must be carefully considered. The CLSTM remains a viable option for more straightforward tasks or environments with limited resources. By addressing their respective weaknesses—such as overfitting and computational efficiency—both models can be further refined to meet the demands of specific use cases.

5.1.3 Analysis of Classical GRU and Quantum GRU

Figures 7 and 8 present the Classical and Quantum GRU model loss and accuracy. The QGRU consistently outperformed the CGRU across all key metrics. QGRU achieved higher training accuracy (~0.790) than CGRU (~0.780), indicating better learning capabilities. Similarly, QGRU demonstrated superior validation accuracy (~0.785) compared to CGRU (~0.775), suggesting better generalization to unseen data. In terms of loss, QGRU also performed better, with lower training loss (~2.7164) and validation loss (~3.0783) compared to CGRU (~2.8513 and ~3.3042, respectively). These results suggest that the quantum-inspired principles employed by QGRU enable more effective feature extraction and pattern recognition, making it better suited for complex code comprehension tasks.

Figure 7: Classical RNN LSTM GRU Model Loss and Accuracy

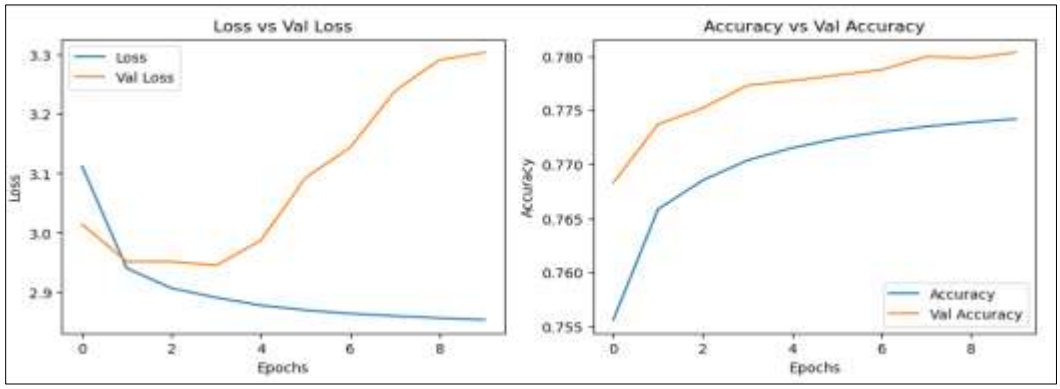
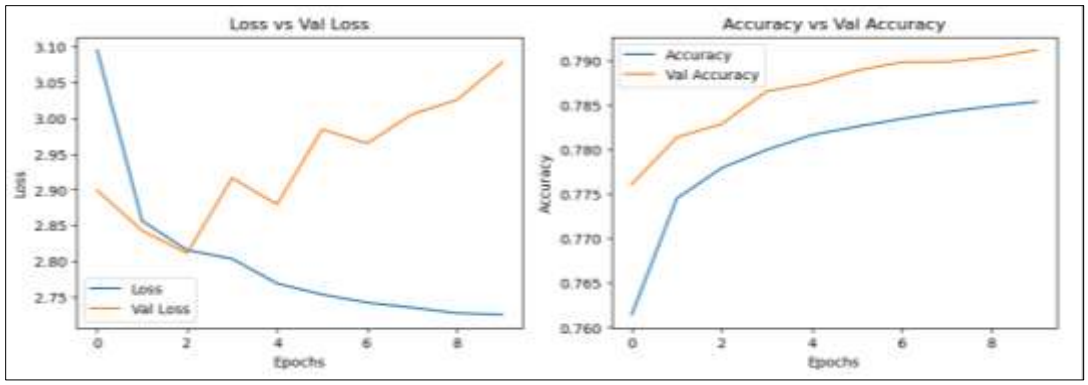


Figure 8: QRNN LSTM GRU Model Loss and Accuracy



However, both models exhibited overfitting, as evidenced by the plateauing or slight increase in validation loss and the stabilization of validation accuracy after a certain number of epochs. This behavior was more pronounced in the CGRU, which struggled to generalize beyond the training data. While QGRU also showed mild overfitting, its impact was less severe, likely due to the model's ability to capture more complex patterns in the data. Both models could benefit from regularization techniques such as dropout, weight decay, or early stopping to mitigate overfitting.

Regarding computational cost, the CGRU has a clear advantage. It is simpler to implement and requires fewer computational resources, making it suitable for resource-constrained environments or simpler tasks. On the other hand, QGRU is more computationally expensive due to its quantum-inspired architecture, which may limit its practicality for large-scale applications without access to specialized hardware. Despite this, QGRU's superior performance makes it a compelling choice for tasks that demand high accuracy and robust generalization.

QGRU performs better than CGRU for code comprehension tasks, achieving higher accuracy and lower loss. However, its higher computational cost and complexity must be carefully considered. By addressing their respective weaknesses—such as overfitting and computational efficiency—both models can be further refined to meet the demands of specific use cases.

5.1.4 Analysis of Classical Bidirectional RNN and Quantum Bidirectional RNN

The CBI-RNN and QBI-RNN models were evaluated for their performance in code comprehension tasks, focusing on training and validation accuracy, loss metrics**,** and overfitting behavior. The results (Figures 9 and 10) reveal significant differences in effectiveness, highlighting each approach's strengths and weaknesses.

Figure 9: Classical Bi-Directional RNN Model Loss and Accuracy

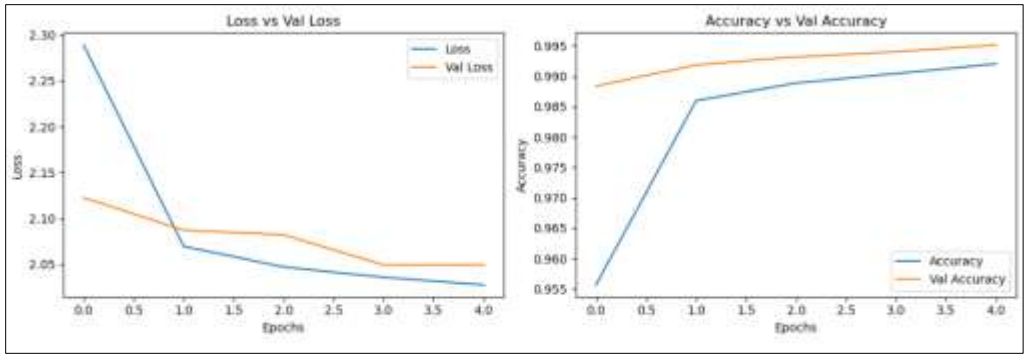
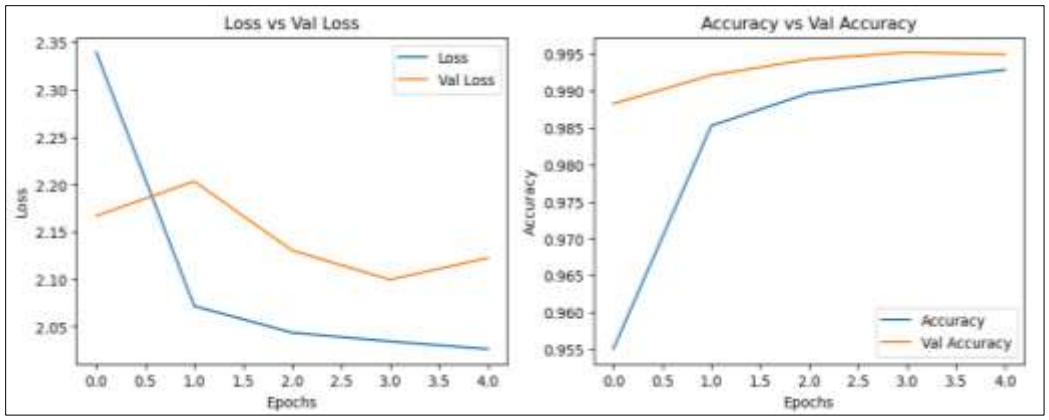


Figure 10: Quantum Bi-Directional RNN Model Loss and Accuracy



The CBI-RNN consistently outperformed the QBI-RNN across all key metrics. CBI-RNN achieved higher training accuracy (~0.995) than QBI-RNN (~0.993), indicating better learning capabilities. Similarly, CBI-RNN demonstrated superior validation accuracy (~0.995) compared to QBI-RNN (~0.993), suggesting better generalization to unseen data. In terms of loss, CBI-RNN also performed better, with lower training loss (~2.0287) and validation loss (~2.0495) compared to QBI-RNN (~2.0241 and ~2.1224, respectively). These results suggest that the classical bidirectional architecture is highly effective for code comprehension tasks, even outperforming quantum-inspired models.

Both models showed signs of overfitting, but this effect was more pronounced in the QBI-RNN. The QBI-RNN's validation loss was slightly higher, while its validation accuracy was marginally lower than the CBI-RNN. This indicates that the QBI-RNN may be picking up noise or specific patterns in the training data that do not generalize well to unseen data.

In terms of computational cost, the CBI-RNN has a clear advantage. It is simpler to implement and requires fewer computational resources, making it suitable for resource-constrained environments or simpler tasks. On the other hand, QRNN is more computationally expensive due to its quantum-inspired architecture, which may limit its practicality for large-scale applications without access to specialized hardware. Despite this, QBI-RNN's potential for capturing complex patterns makes it a compelling choice for tasks that demand high accuracy and robust generalization.

Overall, CBI-RNN performs better than the QBI-RNN for code comprehension tasks, achieving higher accuracy and lower loss. However, its higher computational cost and complexity must be carefully considered. The CBI-RNN remains a viable option for simpler tasks or environments with limited resources. By addressing their respective weaknesses—such as overfitting and computational efficiency—both models can be further refined to meet the demands of specific use cases. Table 2 centralizes the strengths and weaknesses of each tested approach.

Table 2: Strengths and weaknesses of the tested models

Model	Strength	Weakness
CRNN	Simple and easy to implement. Moderate performance for basic code comprehension tasks.	Lower testing accuracy (0.7818) and higher testing loss (3.0025) compared to advanced models. Struggles with long-range dependencies in code.
QRNN	Outperforms CRNN in testing accuracy (0.7948) and testing loss (2.8088). Leverages quantum-inspired principles for better feature extraction.	Slightly higher computational cost compared to classical models. Limited improvement over classical models for this task.

CLSTM	Incorporates LSTM layers to capture long-range dependencies. Moderate testing accuracy (0.7867) and testing loss (2.9625).	Higher training loss (3.2855) compared to QRNN LSTM, indicating less efficient training. Still outperformed by quantum-inspired models.
QRNN LSTM	Combines quantum-inspired principles with LSTM for improved performance. Achieves higher testing accuracy (0.7886) and lower testing loss (2.3223) compared to CLSTM.	Slight overfitting, as testing loss (2.3223) is higher than training loss (2.3089). Requires more computational resources than classical models.
Classical RNN LSTM GRU	Incorporates both LSTM and GRU layers for enhanced sequence modeling. Moderate testing accuracy (0.7803).	Higher testing loss (3.3042) compared to other models, indicating poorer generalization. Outperformed by quantum-inspired models.
QRNN LSTM GRU	Combines quantum-inspired principles with LSTM and GRU layers. Achieves higher testing accuracy (0.7912) compared to Classical RNN LSTM GRU.	Higher testing loss (3.0783) compared to training loss (2.7164), suggesting mild overfitting. More complex and computationally expensive than classical models.
Classical BI Directional RNN	Achieves near-perfect testing accuracy (0.9951) and the lowest testing loss (2.0495). Captures context from both past and future tokens, making it highly effective for code comprehension.	Requires more computational resources than unidirectional models. May be overkill for simpler tasks.
QRNN BI Directional	Combines quantum-inspired principles with bidirectional processing. Achieves the highest training accuracy (0.9932) and near-perfect testing accuracy (0.9950). Lowest training loss (2.0241) among all models.	Slightly higher testing loss (2.1224) compared to Classical BI Directional RNN. Most computationally expensive model.

5.3 Overall analysis of the comparative study

The performance of various code comprehension models was evaluated based on testing accuracy and testing loss. Bidirectional models, particularly CBI-RNN and QBI-RNN, demonstrated superior performance, achieving near-perfect testing accuracy (0.9951 and 0.9950, respectively) and the lowest testing loss (2.0495 and 2.1224), indicating excellent generalization and robustness. QRNN consistently outperformed their classical counterparts, with QRNN and QRNN LSTM achieving higher testing accuracy (0.7948 and 0.7886) and lower testing loss (2.8088 and 2.3223), suggesting that quantum-inspired architectures offer advantages in feature extraction and pattern recognition. However, models like QRNN LSTM GRU showed slight overfitting, as evidenced by higher testing loss (3.0783) than their training loss. In contrast, classical models such as CRNN and Classical RNN LSTM GRU exhibited poorer performance, with lower testing accuracy (0.7818 and 0.7803) and higher testing loss (3.0025 and 3.3042), highlighting the limitations of traditional architectures. Overall, bidirectional and quantum-inspired models emerged as the most effective for code comprehension tasks, with bidirectional RNNs particularly suitable for applications requiring high accuracy and generalization.

6. Conclusion & future work

The comparative analysis reveals that QRNNs, particularly in bidirectional configurations, provide superior performance for code comprehension tasks. Their ability to leverage context and learn from data leads to higher accuracy and lower loss. However, the choice of model depends on the specific requirements of the task, such as the need for high accuracy, generalization, or computational efficiency. The combination of bi-directionality and the capabilities of the QRNN architecture leads to enhanced accuracy and lower loss metrics, making them preferable for tasks involving complex code structures.
As a future work, we will try hybrid architectures by combining quantum models with classical techniques to enhance performance. Also, we aim to train our classical and quantum models on a larger dataset to better evaluate their scalability and generalization. We aim to address overfitting by experimenting with dropout, weight decay, and other regularization techniques and apply data augmentation to generate synthetic training data to improve generalization.

Ethics/Ethical
Not applicable.

Data Availability
The dataset generated and/or analyzed during the current study is available at the GitHub repository for research data using the link:https:/github.com/xing hu/DeepCom. The data that support the findings of this study are available from the corresponding author [Mobeen Alhalabi], upon reasonable request.

Author Contribution Statement
All authors have read and approved the final version of the manuscript. Mobeen W. Alhalabi; data, implementation, writing draft. Fathy E. Eassa; review final draft, supervisior.

Declaration of Funding
The authors declare that no funding was received for this research.

Disclosure of interest
The authors declare that they have no conflicts of interest related to this research.

References

[1] D. a. G. X. a. W. J. a. L. J. a. Z. Z. a. H. E. R. Chen, "Thermodynamic motif analysis for directed stock market networks," Pattern recognition, vol. 114, p. 107872, 2021.

[2] S. a. K. S. A. Bhuvan, "A review of quantum machine learning and discussion of its current status," The Online Journal of Distance Education and e-Learning, vol. 11, 2023.

[3] A. a. K. M. a. A. A. a. L. R.-K. Melnikov, "Quantum machine learning: from physics to software engineering," Advances in Physics: X, vol. 8, p. 2165452, 2023.

[4] A. a. H. K. E. a. V. J.-R. a. M. D. a. O. Y. a. B. P. a. F. A. a. G. A. a. S. L. a. L. D. a. o. Delgado, "Quantum computing for data analysis in high energy physics," arXiv preprint arXiv:2203.08805, 2022.

[5] H. a. C. T. Bez, Quantum computation, Chapman and Hall/CRC, 2023.

[6] D. a. G.-Z. B. a. S.-S. D. Maheshwari, "Quantum machine learning applications in the biomedical domain: A systematic review," Ieee Access, vol. 10, pp. 80463--80484, 2022.

[7] S. a. B. N. J. a. S. T. a. K. N. a. P. P. Pandey, "Quantum machine learning for natural language processing application," Physica A: Statistical Mechanics and its Applications, vol. 627, p. 129123, 2023.

[8] J. Bausch, "Recurrent quantum neural networks," Advances in neural information processing systems, vol. 33, pp. 1368--1379, 2020.

[9] Y. a. W. Z. a. H. R. a. S. S. a. L. J. a. S. R. a. Z. H. a. Z. G. a. G. Y. Li, "Quantum recurrent neural networks for sequential learning," Neural Networks, vol. 166, pp. 148--161, 2023.

[10] M. a. B. A. a. L. S. B. a. S. s. M. Siemaszko, "Rapid training of quantum recurrent neural networks," Quantum Machine Intelligence, vol. 5, p. 31, 2023.

[11] I.-C. a. S. H. a. A. V. L. a. Q. B. a. Y. K. Chen, "A Survey of Classical and Quantum Sequence Models," in 16th International Conference on COMmunication Systems & NETworkS, 2023.

[12] R. a. H. J.-H. a. C. S. Y.-C. a. M. S. a. W. M. Di Sipio, "The dawn of quantum natural language processing," in ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2022.

[13] T. Seidakhmetov, "Question Type Classification Methods Comparison," ArXiv, vol. abs/2001.00571, 2020.

[14] S. V. a. M. A. K. a. K. P. Soman, "A Sequence-Based Machine Comprehension Modeling Using LSTM and GRU," Lecture Notes in Electrical Engineering, 2019.

[15] M. a. M. T. a. I. A. A. a. M. K. S. a. H. M. M. A. a. H. T. a. A. W. U. a. I. A. a. S. R. Hasan, "Codesc: A large code-description parallel dataset," arXiv preprint arXiv:2105.14220, 2021.

[16] Z. a. W. Y. a. P. B. a. C. X. a. S. Z. a. L. Y. a. Y. D. Li, "SeCNN: A semantic CNN parser for code comment generation," Journal of Systems and Software, vol. 181, p. 111036, 2021.

[17] X. L. G. X. X. L. D. &. J. Z. Hu, "Deep code comment generation," in Proceedings of the 26th Conference on Program Comprehension, 2018.