ISSN: 2993-5377

Research article

Open Access Journal of Applied Science and Technology

Quantum Algorithms for Large Language Models on Noisy Intermediate-Scale Quantum Computers

Timo Aukusti Laine*

Financial Physics Lab, Finland

***Corresponding Author**
Timo Aukusti Laine, Financial Physics Lab, Finland.

Submitted: 2026, Mar 02; **Accepted**: 2026, Apr 07; **Published**: 2026, Apr 10

Citation: Laine, T. A. (2026). Quantum Algorithms for Large Language Models on Noisy Intermediate-Scale Quantum Computers. *OA J Applied Sci Technol, 4*(1), 01-13

Abstract

We present a systematic methodology for developing and validating quantum algorithms for Large Language Models (LLMs). This methodology includes partition function-based transformations to map LLM embedding similarity values to a range suitable for quantum computation, and the design of a shallow-circuit quantum algorithm for estimating this transformed similarity measure on nearterm quantum computers. We rigorously evaluate our approach through quantum simulations and experiments on real quantum hardware, demonstrating the feasibility of using quantum computing for LLM embedding analysis. Our results highlight the potential for quantum-inspired techniques in LLM tasks and demonstrate practical strategies for achieving reliable results on noisy quantum hardware.

1. Introduction

Large Language Models (LLMs) have revolutionized natural language processing, fundamentally changing how machines understand and interact with human language. A key component of these models is the representation of textual data as high-dimensional embedding vectors, which capture complex semantic relationships between words, sentences, and documents. Quantum computing, leveraging principles like superposition and entanglement, offers a promising path towards accelerating critical LLM operations and unlocking new capabilities. To realize this potential, new quantum algorithms must be developed that are specifically tailored to the unique characteristics of quantum hardware. This necessitates new approaches to representing and manipulating LLM embeddings within a quantum framework to achieve enhanced performance. However, realizing quantum algorithms for LLMs on current noisy intermediate-scale quantum (NISQ) computers has specific requirements. These requirements include managing the cost of accessing quantum computing resources, addressing the difficulty of characterizing and mitigating noise with limited runs, and developing effective methods to transform LLM data into a format suitable for quantum circuit design, especially when dealing with negative values.

This paper addresses these challenges by introducing a systematic methodology for developing and validating quantum algorithms for LLMs, designed to facilitate progress in the era of quantum computing. Our key contributions are:

- A generalizable, systematic approach applicable to a wide range of quantum simulations.
- An initial problem transformation technique, including linear scaling and partition function-based methods, to create a system more amenable to quantum circuit design.
- A cost-effective validation strategy that begins with a toy example implemented on a quantum simulator, allowing for code verification and algorithm validation without incurring costs.
- Implementation of the same toy example on real quantum hardware to verify code correctness and gain insights into noise levels.
- Application of the validated quantum approach to a realistic LLM embedding problem.

Our methodology provides a rigorous, step-by-step process, starting with the identification of LLM tasks suitable for quantum acceleration and culminating in experimental validation on real quantum hardware. We emphasize practical considerations, such as techniques for ensuring reliable results, at each stage. As a concrete example, we present a partition function-based approach to transform LLM embedding similarity values into a positive range, enabling efficient encoding into quantum states. We then design and implement a shallow-circuit quantum algorithm to estimate this modified similarity measure, rigorously evaluating its performance through quantum simulations and experiments on the `ibm_pittsburgh` quantum computer.

2. Background and Related Work

Large Language Models (LLMs) are driving rapid advancements in natural language processing, enabling breakthroughs in areas such as text generation, machine translation, and question answering [1,2]. These models, often based on the Transformer architecture, represent textual data as high-dimensional embedding vectors, capturing intricate semantic relationships between words, sentences, and entire documents [1,3,4]. Different embedding techniques exist, including contextualized word embeddings from models like BERT and sentence embeddings [5,6].

Training these massive LLMs requires significant computational resources, consuming vast amounts of energy and time. This motivates the exploration of alternative computational paradigms, such as quantum computing, which offers the potential for significant speedups for certain types of computations [7]. Quantum computing harnesses the principles of quantum mechanics, such as superposition and entanglement, to perform computations in a fundamentally different manner than classical computers [7]. Quantum algorithms like Grover's search and quantum annealing have garnered significant attention for their potential to outperform classical counterparts in specific domains [8,9].

The application of quantum computing to LLMs is a nascent but promising field. Lots of work has been made to explore the connections between LLMs and quantum mechanics, drawing inspiration from quantum mechanical concepts to model semantic relationships and develop algorithms for natural language processing. Quantum Natural Language Processing (QNLP) provides a framework for representing language using quantum-inspired structures [10,11].

In previous work, we have explored the application of quantum formalisms to LLM semantic spaces [12-17]. Specifically, in *Semantic Wave Functions: Exploring Meaning in Large Language Models Through Quantum Formalism,* the analogy between LLM embedding spaces and quantum mechanics is explored, positing that LLMs operate within a quantized semantic space where words and phrases behave as quantum states [12]. *The Quantum LLM: Modeling Semantic Spaces with Quantum Principles* clarifies the core assumptions of a quantum-inspired framework for modeling semantic representation and processing in LLMs, providing a detailed exposition of key principles that govern semantic representation, interaction, and dynamics within LLMs [13]. *Quantum LLMs Using Quantum Computing to Analyze and Process Semantic Information* presents a quantum computing approach to analyzing LLM embeddings, leveraging complex-valued representations and modeling semantic relationships using quantum mechanical principles [14]. *Discrete Semantic States and Hamiltonian Dynamics in LLM Embedding Spaces* investigates the structure of LLM embedding spaces using mathematical concepts, particularly linear algebra and the Hamiltonian formalism, drawing inspiration from analogies with quantum mechanical systems [15]. *Quantum Computation of Partition Function Similarity for Large Language Models* introduces a framework for analyzing LLM embedding spaces using partition functions, drawing an analogy to statistical mechanics, and presents an experimental evaluation of its implementation on a quantum computer [16]. *Quantum Hierarchy for Understanding LLM Representations by Modeling Linear Projections and Nonlinear Dynamics* introduces a quantum framework to analyze LLM representations through a layered hierarchy of semantic spaces, modeling phenomena like semantic noise modulation and hallucination emergence using quantum mechanical analogues, and proposes a quantum circuit design for probing LLM embedding spaces [17].

This paper builds upon these foundations by introducing a systematic methodology for developing and validating quantum algorithms for LLMs, with a focus on addressing the challenges of noisy intermediate-scale quantum (NISQ) computers and ensuring the reliability of results through a comprehensive validation process [18]. We employ a partition function-based approach to transform similarity values, drawing inspiration from statistical mechanics, and design a shallow-circuit quantum algorithm for efficient implementation on near-term quantum hardware.

3. Structured Quantum Algorithm Design for Large Language Models

This section details a structured methodology for developing and validating quantum algorithms tailored for Large Language Models (LLMs). The process emphasizes a rigorous, step-by-step approach, designed to facilitate the effective translation of LLM challenges into quantum algorithms suitable for execution on current and near-term quantum computers. This systematic approach is crucial for maximizing the potential of quantum computing in the context of LLMs, while also addressing the realities of noisy intermediate-scale quantum (NISQ) hardware. The key is a process of careful problem selection, transformation, validation, and iterative refinement.

3.1. Problem Identification and Formulation

3.1.1. Identifying Quantum-Advantageous LLM Tasks

The initial step involves identifying specific LLM-related tasks where quantum computing's unique capabilities, particularly its inherent parallelism and potential for algorithmic advances, can be effectively leveraged. The focus should be on problems where quantum algorithms offer a plausible path to improved performance compared to classical methods.

To demonstrate this process, we will use the task of triadic similarity analysis for semantic relationship discovery. This involves quantifying the relationships between LLM embeddings, a fundamental operation in many LLM applications. While this is just one example for illustrative purposes, it highlights the key considerations in selecting a suitable task. The goal is to identify core LLM operations that are computationally intensive and potentially amenable to quantum acceleration.

Beyond embedding similarity analysis, other LLM-related tasks can potentially benefit from quantum-accelerated similarity

calculations. These include document similarity analysis, where similarity scores between document embeddings can reveal thematic relationships; quantum-enhanced recommendation systems, which can identify users with similar tastes by comparing their preference vectors; and quantum anomaly detection, which can flag outlier vectors, indicating anomalous data points, through low similarity scores. The systematic approach outlined here can also be applied to tasks such as optimization of model parameters, generation of text formats, and complex pattern recognition within large text corpora.

3.1.2. Transformation Approaches

A key challenge in applying quantum computing to LLM tasks involving similarity measures is representing these measures, which are often defined over a range of real numbers (including negative values), within the probabilistic framework of quantum computation. Quantum measurements inherently yield positive probabilities, derived from squared amplitudes. Therefore, transforming similarity measures to a positive range suitable for encoding into quantum states is often advantageous for efficient quantum implementation. This step focuses on exploring and applying such transformations.

To illustrate this, we start from the standard definition of cosine similarity

$$S_C = \frac{\mathbf{a} \cdot \mathbf{b}}{||\mathbf{a}||||\mathbf{b}||} \tag{1}$$

We assume vectors **a** and **b** are L2 normalized. This measure, representing the cosine of the angle between the vectors, ranges from -1 to 1.

Transformation 1: Linear Shift

One straightforward approach is a linear shift. Recognizing that

$$S_C = \mathbf{a} \cdot \mathbf{b} = \langle \mathbf{a}|\mathbf{H}|\mathbf{a}\rangle \tag{2}$$

where **H** is a diagonal Hamiltonian with elements $b_i = \lambda_i a_i$, where $\mathbf{a} = [a_1,a_2,...,a_N]$ and $\mathbf{b} = [b_1,b_2,...,b_N]$. We can define a transformed similarity

$$S'_C = \frac{1}{2}(S_C + 1) = \langle \mathbf{a}|\mathbf{H}'|\mathbf{a}\rangle \tag{3}$$

This corresponds to a shifted Hamiltonian, $\mathbf{H'} = (\mathbf{H} + \mathbf{I})/2$, where **I** is the identity matrix, resulting in shifted eigenvalues $\lambda'_i = (\lambda_i + 1)/2$. Consequently,

$$S'_C = \mathbf{a} \cdot \mathbf{b}' \tag{4}$$

where $b'_i = \lambda'_i a_i$. The norm of the transformed vector is

$$||\mathbf{b}'||^2 = \sum_{i=1}^{N} (\lambda' a_i)^2 = \frac{1}{2}(1 + \sum_{i=1}^{N} a_i b_i) = \frac{1}{2}(1 + S_C) \tag{5}$$

The transformed similarity, S_C', is guaranteed to be positive and lie within the range of 0 to 1, making it suitable for amplitude encoding in a quantum circuit. An interesting detail between **b** and **b′** is that

$$||\mathbf{b}||^2 = \sum_{i=1}^{N} b_i^2 = 1 \tag{6}$$

$$||\mathbf{b}'||^2 = \sum_{i=1}^{N} b_i'^2 = S'_C \in [0,1] \tag{7}$$

Vector **b** is L2 normalized and represents a normalized quantum state, while the squared norm of **b′** is equal to S_C', which lies between 0 and 1.

Transformation 2: Partition Function-Based Similarity

As an alternative, we assume that the components of the embedding vectors, a_i and b_i, can be interpreted as eigenvalues representing the energy levels of a system. This eigenvalue interpretation naturally

leads to the use of the partition function from statistical mechanics. Given this interpretation, we define the partition functions for vectors **a** and **b** as

$$Z(\mathbf{a}) = \sum_{i=1}^{N} \exp(-\beta a_i) \tag{8}$$

$$Z(\mathbf{b}) = \sum_{i=1}^{N} \exp(-\beta b_i) \tag{9}$$

where β is a positive parameter analogous to the inverse temperature in statistical mechanics. We then define a similarity measure based on these partition functions

$$S_Z = \sum_{i=1}^{N} Z(\mathbf{a})_i Z(\mathbf{b})_i \tag{10}$$

The key idea is that minimizing S_Z corresponds to finding the most similar vectors in this transformed space. This stems from the exponential transformation $\exp(-\beta x)$, which assigns smaller values to larger x. Consequently, vectors sharing aligned positive components (e.g., $a_i = 1, b_i = 1$) will yield a smaller S_Z, reflecting a notion of similarity based on shared "low-energy" states. Conversely, aligned negative components (e.g., $a_i = -1, b_i = -1$) result in a larger S_Z as the partition function framework interprets shared absence or opposition as contributing to dissimilarity.

However, S_Z differs from cosine similarity in a crucial way: it is sensitive not only to the relative alignment of vectors but also to their absolute position in the embedding space. For instance, identical vectors with components equal to +0.9 will yield a much smaller S_Z than identical vectors with components equal to −0.9, even though both have a cosine similarity of 1. This is because S_Z is rooted in a physical interpretation where embedding components represent energy levels, not just directional relationships.

The question is: what does a negative value in an embedding dimension mean? Cosine similarity implicitly assumes a directional relationship, where negative values represent opposing directions. However, if a negative value signifies the absence of a feature or an opposite characteristic, then the partition function approach offers a more nuanced perspective. In this case, two negative values indicate a shared absence or opposition, which decreases similarity, consistent with maximizing S_Z. The partition function transformation provides a physical interpretation, mapping embedding vector components to energy levels in a statistical mechanical system. Therefore, cosine similarity and partition function-based similarity offer distinct perspectives on similarity, and their suitability depends on the specific context and the desired interpretation of the embedding vector components.

Experimentally determining which measure is "better" is challenging. For example, in Retrieval-Augmented Generation (RAG) systems, both methods typically return the top k results, but there is no definitive ground truth to assess which results are truly the most relevant. The choice of similarity measure ultimately depends on the specific application and the desired semantic properties.

Transformation 3: Linear Shift + Partition Function

To combine the benefits of both approaches and improve the numerical stability and performance of the quantum circuit, we can apply a linear shift before the partition function transformation. This ensures that the input to the exponential function is always positive.

We define a shifted vector $\tilde{\mathbf{a}}$ such that $\tilde{a}_i = a_i + C$, where C is a constant chosen to make all components positive. A simple choice is C = 1, assuming that the components of a are greater than -1. We can then define the partition function using the shifted vector

$$Z(\tilde{\mathbf{a}}) = Z(\mathbf{a}, C) = \sum_i \exp(-\beta \tilde{a}_i) = \sum_i \exp(-\beta(a_i + C)) \tag{11}$$

The similarity measure is then

$$S_{Z,C} = \sum_{i=1}^{N} Z(\mathbf{a}, C)_i Z(\mathbf{b}, C)_i = \sum_{i=1}^{N} \exp(-\beta(\tilde{a}_i + \tilde{b}_i)) = \sum_{i=1}^{N} \exp(-\beta(a_i + b_i + 2C)) \tag{12}$$

If we choose $C = 1$ and β is appropriately scaled, $S_{Z,C}$ can be constrained to be between 0 and 1. This combined approach leverages the physical interpretation of the partition function while ensuring that the values are suitable for quantum computation. This linear shift effectively translates the range of the input vector components. While it alters the absolute magnitude of the partition function values, it crucially preserves the relative order of similarity between different vector combinations. This means that if one set of vectors is more similar than another before the shift, their relative similarity ranking remains consistent after the shift. This preservation of relative order is often the primary concern in tasks like similarity search or clustering. For example, in the next section, when we discuss triadic similarity, this property ensures that the ranking of clustered similarities remains meaningful. Therefore, the practical advantages for quantum circuit implementation, such as ensuring positive amplitudes for encoding, often outweigh the consideration of altering the absolute values of the underlying system components in this context.

3.1.3. Applying Transformations to Triadic Similarity

In this step, we apply the transformations discussed earlier to the specific task of calculating triadic similarity, which is relevant to applications like clustered embedding analysis. We aim to obtain a similarity measure that is both meaningful and easily implementable in a quantum circuit.

We start with the standard definition of cosine similarity for three vectors, known as triadic cosine similarity

$$S_C = \frac{(\mathbf{a}\cdot\mathbf{b})(\mathbf{a}\cdot\mathbf{c})(\mathbf{b}\cdot\mathbf{c})}{||\mathbf{a}||||\mathbf{b}||||\mathbf{c}||} \tag{13}$$

Assuming vectors **a, b**, and **c** are L2 normalized, this simplifies to

$$S_C = (\mathbf{a}\cdot\mathbf{b})(\mathbf{a}\cdot\mathbf{c})(\mathbf{b}\cdot\mathbf{c}) \tag{14}$$

This measure captures the product of the pairwise cosine similarities. Applying quantum computation to similarity measures like cosine similarity presents challenges due to the presence of negative values. Quantum measurements inherently yield positive probabilities, derived from squared amplitudes. Therefore, transforming the similarity measure to a positive range can be advantageous for quantum implementation.

Using the partition function approach with linear scaling, and to mimic the structure of the triadic cosine similarity, we define the triadic partition function similarity, $S_{Z,triadic,}$ as the product of pairwise partition function similarities

$$S_{Z,triadic} = S_{Z,ab}S_{Z,ac}S_{Z,bc} \tag{15}$$

$$= \left(\sum_{i=1}^{N}\exp(-\beta(a_i+b_i+2C))\right)\left(\sum_{i=1}^{N}\exp(-\beta(a_i+c_i+2C))\right)\left(\sum_{i=1}^{N}\exp(-\beta(b_i+c_i+2C))\right) \tag{16}$$

Alternatively, because we aim to find a minimum for $S_{Z,triadic}$, we can define a simpler measure by taking the product inside the summation

$$S_{Z,alt} = \sum_{i=1}^{N} Z(\mathbf{a})_i Z(\mathbf{b})_i Z(\mathbf{c})_i = \sum_{i=1}^{N}\exp(-\beta(a_i+b_i+c_i+3C)) \tag{17}$$

The key idea is that minimizing either $S_{Z,triadic}$ or $S_{Z,alt}$ corresponds to finding the most similar vectors in this transformed space. This is because the exponential transformation exp(−βx) yields a smaller value for larger x. Consequently, vectors with many aligned positive components will lead to a smaller $S_{Z,}$ indicating similarity. Conversely, vectors with many aligned negative components will lead to a larger $S_{Z,}$ indicating dissimilarity.

While the form of $S_{Z,alt}$ might superficially resemble the expression $\sum a_i b_i c_i$, the latter is problematic because it attempts to directly extend the concept of cosine similarity (which measures the angle between two vectors) to a three-vector relationship without a clear geometric or physical justification. The direct product lacks a well-defined interpretation in terms of vector relationships, and simply summing these products doesn't provide a meaningful measure of similarity. In contrast, $S_{Z,alt}$ uses the exponential transformation before the summation, mapping the original vector components to a space where negative values (representing, for example, the absence of a feature) are treated differently, allowing it to capture a notion of similarity based on shared "low-energy" states.

3.2. Quantum Circuit Design

3.2.1. Quantum Circuit Design for Similarity Estimation

Having chosen the partition function-based similarity measure and the triadic similarity example, we now design a quantum circuit to estimate this measure. To estimate the similarity between N-dimensional embedding vectors **a, b,** and **c** using our partition function-based approach, we employ a quantum circuit consisting of three sets of N qubits, as illustrated in Figure 1. Each set of N qubits represents one of the vectors. While this circuit is designed for three vectors, the same principle can be extended to other similarity measures and to varying numbers of vectors. We denote $\tilde{\mathbf{a}} = \mathbf{a} + 1$, $\tilde{\mathbf{b}} = \mathbf{b} + 1$, and $\tilde{\mathbf{c}} = \mathbf{c} + 1$. This circuit design demonstrates the encoding and measurement of individual terms on a quantum computer, but the summation across all *N* dimensions is still performed classically.

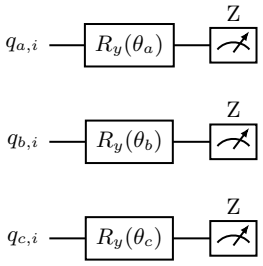

Figure 1: Simplified Quantum Circuit for Estimating Similarity between Partition Functions. The Qubits $q_{a,i}$, $q_{b,i}$, and $q_{c,i}$ Encode the *i*-th Components of Vectors **a, b,** and **c,** Respectively, for a Single Dimension *i.*

For each dimension *i,* the corresponding qubits $q_{a,i}$, $q_{b,i,}$ and $q_{c,i}$ are initialized using a R_y rotation. Specifically, $q_{a,i}$ is prepared in the state $A_{1i}|0\rangle + A_{2i}|1\rangle$, where $A_{1i} = Z(\tilde{\mathbf{a}})_i$ and $A_{2i} = \sqrt{1 - A_{1i}^2}$. Similarly, $q_{b,i}$ is prepared in the state $B_{1i}|0\rangle + B_{2i}|1\rangle$, where $B_{1i} = Z(\tilde{\mathbf{b}})_i$ and $B_{2i} = \sqrt{1 - B_{1i}^2}$, and $q_{c,i}$ is prepared in the state $C_{1i}|0\rangle + C_{2i}|1\rangle$, where $C_{1i} = Z(\tilde{\mathbf{c}})_i$ and $C_{2i} = \sqrt{1 - C_{1i}^2}$. Here, $Z(\tilde{\mathbf{a}})_i = \exp(-\beta\tilde{a}_i)$, $Z(\tilde{\mathbf{b}})_i = \exp(-\beta\tilde{b}_i)$, and $Z(\tilde{\mathbf{c}})_i = \exp(-\beta\tilde{c}_i)$, with $\tilde{\mathbf{a}}_i = a_i + 1$, $\tilde{\mathbf{b}}_i = b_i + 1$, and $\tilde{\mathbf{c}}_i = c_i + 1$.

The angles θ_a, θ_b, and θ_c of the R_y gates are chosen such that $A_{1i} = \cos(\theta_a/2)$, $B_{1i} = \cos(\theta_b/2)$, and $C_{1i} = \cos(\theta_c/2)$. This encoding maps the normalized partition function value for each dimension to the amplitude of the $|0\rangle$ state. This approach is known as amplitude encoding. We choose amplitude encoding for its simplicity and ease of implementation. While other encoding methods, such as phase encoding, are possible, amplitude encoding allows us to directly represent the transformed vector components as probabilities. An important point with this parameter choice is that the normalization is automatically satisfied. If $a_i = [-1,1]$, then $\tilde{\mathbf{a}}_i = [0,2]$ and $Z(\tilde{\mathbf{a}}_i) = [e^{-1},1]$, and therefore $Z(\tilde{\mathbf{a}}_i)$ is directly applicable in a quantum circuit.

After initialization, each qubit is measured in the *Z*-basis. The tensor product creates a multi-qubit state where the amplitudes represent the joint probabilities of measuring each qubit in a particular state. Consider the tensor product of the qubits representing dimension *i*

$$|\psi_i\rangle = (A_{1i}|0\rangle + A_{2i}|1\rangle) \otimes (B_{1i}|0\rangle + B_{2i}|1\rangle) \otimes (C_{1i}|0\rangle + C_{2i}|1\rangle) \tag{18}$$

$$\begin{aligned} =& A_{1i}B_{1i}C_{1i}|000\rangle + A_{1i}B_{1i}C_{2i}|001\rangle + A_{1i}B_{2i}C_{1i}|010\rangle + A_{1i}B_{2i}C_{2i}|011\rangle \\ &+ A_{2i}B_{1i}C_{1i}|100\rangle + A_{2i}B_{1i}C_{2i}|101\rangle + A_{2i}B_{2i}C_{1i}|110\rangle + A_{2i}B_{2i}C_{2i}|111\rangle \end{aligned} \tag{19}$$

The probability of measuring the state $|000\rangle$ for dimension *i* is

$$P_i(|000\rangle) = (A_{1i}B_{1i}C_{1i})^2 = \left[\exp(-\beta(a_i + b_i + c_i + 3))\right]^2 \tag{20}$$

where $C = 1$. The tensor product and subsequent measurement of the $|000\rangle$ state effectively calculates the product of the normalized partition functions for each dimension. By summing these probabilities across all dimensions, we obtain our final similarity measure

$$S_Z = \sum_{i=1}^{N} P_i(|000\rangle) = \sum_{i=1}^{N} \exp(-2\beta(a_i + b_i + c_i + 3)) \tag{21}$$

Note that with this circuit design, the parameter β in Eq. (21) is effectively doubled. For simplicity of comparison with the theory, we select $\beta = 1/2$ to compensate this effect and get

$$S_Z = \sum_{i=1}^{N} P_i(|000\rangle) = \sum_{i=1}^{N} \exp(-(a_i + b_i + c_i + 3)) \tag{22}$$

This circuit provides a simple and straightforward approach to estimating the similarity between LLM embeddings using quantum computation. This design, while relatively basic, serves as a foundation for exploring more complex quantum algorithms for LLM analysis.

3.3. Experimental Validation

3.3.1. Quantum Simulator Validation with a Toy Example

Before deploying our quantum circuit on real quantum hardware, a crucial step is to validate its theoretical correctness and implementation accuracy using a quantum simulator. While publicly available quantum computers boast an increasing number of qubits (up to 156 or more), their results can be significantly affected by noise and decoherence. Furthermore, the high cost of using these resources limits the ability to perform extensive troubleshooting and repeated trials. Therefore, we employ a quantum simulator to test our circuit in an idealized, noise-free environment. This allows us to verify the correctness of the quantum algorithm's implementation in a fast and cost-efficient manner. Subsequent experiments on real quantum hardware then address a different research question: namely, how much noise affects the computation and whether the fidelity of current quantum hardware is sufficient to reproduce the results obtained with the simulator.

Our testing strategy involves creating a "toy example" with low-dimensional embedding vectors. This toy example is designed to be simple enough to verify by hand, allowing for a direct comparison between theoretical calculations and simulation results. Specifically, in this case we consider three vectors with dimension $N = 3$. This results in a quantum circuit with only 9 qubits, a size easily handled by most quantum simulators. By using a quantum simulator, we can verify the correctness of our quantum circuit design, the accuracy of our code implementation, and the proper initialization of the quantum state.

We select the following arbitrary three vectors:

$$a = [0.3, 0.4, \sqrt{1 - 0.3^2 - 0.4^2}] \tag{23}$$

$$b = [0.2, 0.5, \sqrt{1 - 0.2^2 - 0.5^2}] \tag{24}$$

$$c = [0.1, 0.6, \sqrt{1 - 0.1^2 - 0.6^2}] \tag{25}$$

These vectors were chosen to be L2 normalized and to have a mix of positive values to test the behavior of the partition function-based similarity measure.

The theoretical value, calculated using the exact cluster similarity method (see Appendix A1), is

$$S_{Z,alt,9} = 0.0425 \tag{26}$$

The quantum circuit calculation was performed using the Qiskit code shown in Appendix A2. The simulation, with 8192 shots, gives

$$S_{Z,simulator,9} = 0.0438 \tag{27}$$

The relative error, calculated as $|S_{\text{theoretical}} - S_{\text{simulated}}|/S_{\text{theoretical}}$, is approximately 3%. We can also study the effect of changing the number of shots. For example, in our example, if lowering the number of shots to 4096, the relative error is greater than 3%, and when it is, for example, 16384, the relative error is closer to 2%.

This step allows us to isolate and eliminate potential errors in our theoretical understanding or code before proceeding to more expensive and complex experiments on real quantum hardware.

3.3.2. Real Quantum Hardware Validation with the Toy Example

Having validated our quantum circuit and code implementation using a quantum simulator, we now transition to testing on real quantum hardware. This step is crucial for assessing the impact of noise and decoherence on the accuracy of our results and for verifying the code on a real quantum platform. Due to the high cost of using real quantum computing resources, we recommend a two-part workflow: first, execute the quantum computation and save the raw results to a file; second, perform the result analysis in a separate script. This modular approach ensures that the quantum computation needs to be executed only once, optimizing the utilization of these comparatively expensive services.

We implement the same "toy example" circuit with $N = 3$ (resulting in 9 qubits) on a publicly available quantum computer

using the IBM Quantum cloud service. Our script selects the least busy backend, which in this case was `ibm_pittsburgh` due to its availability and sufficient number of qubits. The code for running the quantum circuit and saving the results is provided in Appendix A3. The code for analyzing the saved results and calculating the triadic similarity is provided in Appendix A4. Note that the method for accessing the results from the quantum computer differs from the method used for the quantum simulator, as the real quantum computer requires accessing the results from the saved file.

We now use the same initial vectors Eqs. (23)-(25). When making the triadic similarity calculation with 8192 shots, and analyzing the results using the script in Appendix A4, we obtain the result

$$S_{Z,pittsburgh,9} = 0.0455 \tag{28}$$

The relative error is approximately 7%. While further reruns could quantify the fluctuation of this relative error, this was not performed due to the high cost associated with service usage. The results indicate that the script functions correctly, with additional error attributable to hardware noise and decoherence effects.

3.3.3. Scaling to Realistic LLM Embeddings on Quantum Hardware

Having validated our approach with a toy example and characterized the performance of our quantum circuit on real hardware, we now scale up to a more realistic scenario using LLM embedding vectors. We consider three embedding vectors, **a, b,** and **c,** each with a dimension of $N = 32$. This requires a quantum circuit with a total of 96 qubits. The code used to generate these embeddings is detailed in Appendix A5.

Scaling quantum algorithms to larger problem sizes presents several challenges, including increased qubit requirements and the accumulation of noise. To address these challenges, we employ strategies such as circuit optimization, hardware-aware compilation, and the potential for error mitigation techniques.

For this experiment, we use the OpenAI LLM model and 32-dimensional embeddings. OpenAI allows for the creation of 32-dimensional embeddings using their third-generation models through a feature called Matryoshka Representation Learning. Matryoshka Representation Learning allows for generating embeddings of varying dimensionality from a single model, which is useful for adapting to the limited number of qubits available on current quantum hardware. We select three sentences for this experiment:

1. Sentence: ”The cat sat quietly on the warm windowsill watching birds outside.”
2. Sentence: ”A dog ran joyfully through the muddy puddles in the backyard.”
3. Sentence: ”The ancient oak tree stood tall against the stormy evening sky.”

The sentences are chosen such that the first two (cat and dog) are semantically closer, both describing domestic animals engaged in everyday activities. The third sentence (oak tree) is semantically more distant, focusing on nature and weather. This selection provides a meaningful test where the calculated triadic similarity is expected to reflect the underlying semantic relationships between the sentences.

We execute the quantum script using the backend `ibm_pittsburgh` with 8192 shots. The theoretical result is

$$S_{Z,alt,96} = 1.616 \tag{29}$$

and the quantum computer result is

$$S_{Z,pittsburgh,96} = 1.737 \tag{30}$$

The relative error is 7.5%. Interestingly, the error in this 96-qubit calculation was not significantly larger than in the 9-qubit case. This suggests that, for this specific circuit design and on the ibm pittsburgh backend, the effects of noise do not increase dramatically with the number of qubits. This could be due to factors such as the relatively shallow circuit depth and the specific noise characteristics of the hardware.
These results demonstrate the feasibility of scaling our approach to more realistic LLM embedding sizes on current quantum hardware.

3.4. Error Mitigation

3.4.1. Error Mitigation Strategies

The execution of quantum algorithms on noisy intermediate-scale quantum (NISQ) hardware inevitably introduces errors that can degrade the accuracy of results. To obtain reliable results from quantum computations, particularly as circuit depth and qubit count increase, error mitigation techniques offer the potential to significantly improve accuracy. Unlike full quantum error correction, which requires a large overhead of physical qubits, error mitigation aims to reduce the impact of noise on expectation values using classical post-processing or minor modifications to the quantum circuit, without requiring fault-tolerant hardware.

Several programmatic approaches exist for mitigating errors. Readout error mitigation addresses errors that occur during the measurement process by characterizing measurement errors

and statistically correcting for them. Zero-Noise extrapolation (ZNE) involves running the quantum circuit at several artificially increased noise levels and then extrapolating to the theoretical zero-noise limit. Other techniques, such as dynamical decoupling, can also be used to suppress decoherence.

While these error mitigation techniques offer the potential to extract meaningful signals from NISQ devices, their application often incurs additional computational cost, either in terms of increased quantum circuit executions or classical post-processing. Due to the high cost associated with extensive quantum hardware usage, the present work focuses on demonstrating the feasibility of the proposed quantum algorithm and characterizing its performance without explicit error mitigation.

4. Conclusions

This work has presented a systematic methodology that streamlines the development and validation of quantum algorithms tailored for Large Language Models (LLMs), providing a practical roadmap for navigating the complexities of current quantum hardware and accelerating the exploration of quantum-enhanced LLM applications. Our approach is structured into distinct steps, beginning with the identification of LLM tasks where quantum computing can offer a potential advantage. This is followed by the crucial phase of transforming the problem into a format suitable for quantum computation, prioritizing circuit simplicity for efficient deployment on noisy intermediate-scale quantum (NISQ) devices. We introduced a new initial problem transformation technique, including linear scaling and partition function-based methods, to create a system more amenable to quantum circuit design.

A key aspect of our methodology involves a phased validation process. We first employ a toy model, rigorously tested on quantum simulators, to verify the theoretical correctness of the quantum circuit design and the accuracy of the code implementation. This allows us to gain an initial understanding of the expected error levels in an idealized environment. Subsequently, the same toy model is deployed on real quantum hardware. This step is vital for code verification on specific quantum platforms and for characterizing the impact of real-world noise and decoherence on the results. Once validated, the methodology guides the scaling of the quantum algorithm to address realistic LLM embedding problems.

A practical finding from our hardware experiments highlights the importance of a two-step computational strategy. To optimize the utilization of currently expensive quantum computing resources, we recommend separating the workflow: first, execute the quantum initialization and computation, saving the raw results; second, perform all subsequent analysis classically from the saved data. This minimizes the quantum processing unit (QPU) time, allowing for more extensive classical analysis without incurring repeated QPU costs.

Our results, demonstrating a consistent relative error of approximately 7% between quantum simulations and hardware executions for both small (9-qubit) and larger-scale (96-qubit) problems on the ibm pittsburgh backend, confirm the feasibility and effectiveness of this systematic approach. Importantly, we emphasize that the initial quantum simulator runs provide a fast and cost-effective way to verify the correctness of the algorithm's implementation, while the subsequent hardware experiments serve to investigate the impact of noise and assess the fidelity of current quantum devices. This consistency suggests that, for the shallow-circuit design employed, the effects of noise do not significantly increase with qubit count on the tested hardware. Depending on the achieved accuracy and the specific application requirements, further enhancements can be pursued through the application of various error mitigation techniques, such as readout error mitigation or zero-noise extrapolation. This systematic framework provides a generalizable blueprint for tackling a wide array of LLM-related problems at the intersection of quantum computing and artificial intelligence, enabling the development of more powerful and efficient quantum-enhanced LLMs. Beyond the technical advancements, this work contributes to the broader effort of developing more efficient and sustainable AI technologies. Exploring quantum-enhanced LLMs allows us to aim to reduce the computational resources required for natural language processing, potentially leading to lower energy consumption and a reduced environmental footprint for AI applications.

Appendix A: Code Listings

1. Exact Cluster Similarity Calculation Code

Listing 1: Python code for calculating the exact cluster similarity.

```
import numpy as np

def exact_cluster_similarity(a, b, c, beta):
    """Calculates␣the␣exact␣cluster␣similarity␣using␣the␣formula."""
    sz = 0
    for i in range(len(a)):
        sz += np.exp(-2 * beta * (a[i] + b[i] + c[i] + 3))
    return sz
```

2. Quantum Simulator Code

Listing 2: Qiskit code for quantum triadic similarity calculation using a quantum simulator.

```python
import numpy as np
from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator

def quantum_simulator_triadic_similarity(a, b, c, beta, shots):
    """Calculates the triadic similarity using a quantum simulator with a single circuit."""
    # Error handling: Check if input vectors have the same length
    if not (len(a) == len(b) == len(c)):
        raise ValueError("Input vectors must have the same length.")

    num_dimensions = len(a)
    num_qubits = num_dimensions * 3

    # Create a single circuit for all dimensions
    qc = QuantumCircuit(num_qubits, num_qubits)

    for i in range(num_dimensions):
        # Calculate partition function values (shifted)
        a_tilde = a[i] + 1
        b_tilde = b[i] + 1
        c_tilde = c[i] + 1

        partition_a = np.exp(-beta * a_tilde)
        partition_b = np.exp(-beta * b_tilde)
        partition_c = np.exp(-beta * c_tilde)

        # Calculate rotation angles
        theta_a = 2 * np.arccos(np.clip(partition_a, -1, 1))
        theta_b = 2 * np.arccos(np.clip(partition_b, -1, 1))
        theta_c = 2 * np.arccos(np.clip(partition_c, -1, 1))

        # Apply Ry rotations to qubits for this dimension
        # Dimension i uses qubits: 3*i, 3*i+1, 3*i+2
        qc.ry(theta_a, 3*i)
        qc.ry(theta_b, 3*i + 1)
        qc.ry(theta_c, 3*i + 2)

    # Measure all qubits
    qc.measure(range(num_qubits), range(num_qubits))

    # Simulate the circuit once
    simulator = AerSimulator()
    compiled_circuit = transpile(qc, simulator)
    job = simulator.run(compiled_circuit, shots=shots)
    result = job.result()
    counts = result.get_counts(compiled_circuit)

    # Calculate sz by summing probabilities where each dimension's 3 qubits are |000>
    sz = 0
    for bitstring, count in counts.items():
        # Count how many dimensions have all three qubits as |0>
        # Note: Qiskit bitstrings are reversed (right-to-left)
        for i in range(num_dimensions):
            # Check if qubits 3*i, 3*i+1, 3*i+2 are all '0'
            # In reversed string: positions -(3*i+1), -(3*i+2), -(3*i+3)
            if (len(bitstring) > 3*i+2 and
                bitstring[-(3*i+1)] == '0' and
                bitstring[-(3*i+2)] == '0' and
                bitstring[-(3*i+3)] == '0'):
                sz += count / shots

    return sz
```

3. Real Hardware Code (Part 1): Quantum Computation and Result Saving

Listing 3: Qiskit code for quantum triadic similarity calculation on real quantum hardware - quantum computation and result saving.

```python
import numpy as np
from qiskit import QuantumCircuit, transpile
from qiskit_ibm_runtime import QiskitRuntimeService, Session, Sampler
import pickle

def quantum_computer_triadic_similarity_run(a, b, c, beta, shots=4096, filename='
    triadic_similarity_results.pkl'):
    """
␣␣␣␣Creates␣a␣quantum␣circuit␣for␣triadic␣similarity␣and␣runs␣it␣on␣a␣real␣quantum␣computer.
␣␣␣␣Stores␣the␣raw␣result␣in␣a␣pickle␣file.␣No␣result␣analysis␣is␣done␣here.
␣␣␣␣"""
    num_dimensions = len(a)
    num_qubits = num_dimensions * 3

    qc = QuantumCircuit(num_qubits, num_qubits)

    for i in range(num_dimensions):
        a_tilde = a[i] + 1
        b_tilde = b[i] + 1
        c_tilde = c[i] + 1

        partition_a = np.exp(-beta * a_tilde)
        partition_b = np.exp(-beta * b_tilde)
        partition_c = np.exp(-beta * c_tilde)

        theta_a = 2 * np.arccos(np.clip(partition_a, -1, 1))
        theta_b = 2 * np.arccos(np.clip(partition_b, -1, 1))
        theta_c = 2 * np.arccos(np.clip(partition_c, -1, 1))

        qc.ry(theta_a, 3 * i)
        qc.ry(theta_b, 3 * i + 1)
        qc.ry(theta_c, 3 * i + 2)

    qc.measure(range(num_qubits), range(num_qubits))

    service = QiskitRuntimeService()
    backend = service.least_busy(simulator=False, operational=True, min_num_qubits=num_qubits)
    print(f"Using␣backend:␣{backend.name}")

    transpiled_qc = transpile(qc, backend)

    with Session(backend=backend) as session:
        sampler = Sampler()
        job = sampler.run([transpiled_qc], shots=shots)
        print(f"Job␣ID:␣{job.job_id()}")
        result = job.result()

        with open(filename, 'wb') as f:
            pickle.dump(result, f)

        print(f"Results␣stored␣in␣{filename}")
```

4. Real Hardware Code (Part 2): Result Analysis

Listing 4: Qiskit code for quantum triadic similarity calculation on real quantum hardware - result analysis.

```python
import pickle
import numpy as np
from collections import Counter

def quantum_computer_triadic_similarity_analysis(filename='triadic_similarity_results.pkl'):
```

```
    """
␣␣␣␣Analyzes␣the␣raw␣results␣from␣the␣quantum␣computer␣calculation␣and␣calculates␣the␣triadic␣
    similarity.
␣␣␣␣"""
    try:
        with open(filename, 'rb') as f:
            result = pickle.load(f)
    except FileNotFoundError:
        print(f"Error:␣File␣'{filename}'␣not␣found.")
        return None

    pub_result = result[0]
    bit_array = pub_result.data['c']
    bitstrings = bit_array.get_bitstrings()
    counts = Counter(bitstrings)
    total_shots = sum(counts.values())

    num_qubits = len(bitstrings[0])
    num_dimensions = num_qubits // 3

    sz = 0.0
    for i in range(num_dimensions):
        for bitstring, count in counts.items():
            q0 = len(bitstring) - 1 - (3 * i)
            q1 = len(bitstring) - 1 - (3 * i + 1)
            q2 = len(bitstring) - 1 - (3 * i + 2)
            if bitstring[q0] == '0' and bitstring[q1] == '0' and bitstring[q2] == '0':
                sz += count / total_shots

    print(f"Number␣of␣dimensions:␣{num_dimensions}")
    print(f"Total␣shots:␣{total_shots}")
    print(f"Calculated␣sz:␣{sz}")

    return sz
```

5. OpenAI Embedding Generation Code

Listing 5: Python code for generating LLM embeddings using the OpenAI API.

```
from openai import OpenAI

client = OpenAI()

sentences = [
    "The␣cat␣sat␣quietly␣on␣the␣warm␣windowsill␣watching␣birds␣outside.",
    "A␣dog␣ran␣joyfully␣through␣the␣muddy␣puddles␣in␣the␣backyard.",
    "The␣ancient␣oak␣tree␣stood␣tall␣against␣the␣stormy␣evening␣sky."
]

embeddings = []
for sentence in sentences:
    response = client.embeddings.create(
        input=sentence,
        model="text-embedding-3-small",
        dimensions=32
    )
    embeddings.append(response.data[0].embedding)

a = embeddings[0]
b = embeddings[1]
c = embeddings[2]
```

Reference:

1. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems, 30.*
2. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. *Advances in neural information processing systems, 33,* 1877-1901.
3. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space.

International Conference on Learning Representations.

4. Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing* (EMNLP) (pp. 1532-1543).
5. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019, June). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)* (pp. 4171-4186).
6. Reimers, N., & Gurevych, I. (2019, November). Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)* (pp. 3982-3992).
7. Nielsen, M. A., & Chuang, I. L. (2010). Quantum computation and quantum information. *Cambridge university press.*
8. Grover, L. K. (1996, July). *A fast quantum mechanical algorithm for database search. In Proceedings of the twenty-eighth annual ACM symposium on Theory of computing* (pp. 212-219).
9. Kadowaki, T., & Nishimori, H. (1998). Quantum annealing in the transverse Ising model. *Physical Review E, 58*(5), 5355.
10. Coecke, B., Sadrzadeh, M., & Clark, S. (2010). Mathematical foundations for a compositional distributional model of meaning. *Lambek Festschirft, Special Issue of Linguistic Analysis,* abs/1003.4394.
11. Widdows, D., & Widdows, D. (2004). *Geometry and meaning* (Vol. 773). *Stanford: CSLI publications.*
12. Laine, T. A. (2025). Semantic Wave Functions: Exploring Meaning in Large Language Models Through Quantum Formalism. *OA J Applied Sci Technol, 3*(1), 01-22.
13. Laine, T. A. (2025). The Quantum LLM: Modeling Semantic Spaces with Quantum Principles. *OA J Applied Sci Technol, 3*(2), 01-13.
14. Laine, T. A. (2026). Quantum LLMs Using Quantum Computing to Analyze and Process Semantic Information. *OA J Applied Sci Technol, 4*(1), 01-19.
15. Laine, T. A. (2026). Discrete Semantic States and Hamiltonian Dynamics in LLM Embedding Spaces. *OA J Applied Sci Technol, 4*(1), 01-23.
16. Laine, T. A. (2026). Quantum Computation of Partition Function Similarity for Large Language Models. *OA J Applied Sci Technol, 4*(1), 01-23.
17. Laine, T. A. (2026). Quantum Hierarchy for Understanding LLM Representations by Modeling Linear Projections and Nonlinear Dynamics. *OA J Applied Sci Technol, 4*(1), 01-43.
18. Preskill, J. (2018). Quantum computing in the NISQ era and beyond. *Quantum, 2*, 79.