Quantum Matrix Multiplication and Effects on Large Language Models

Aditya Gollamudi
Department of Computer Science
Texas A&M University
College Station, Texas
adigo@tamu.edu

***Abstract*—Matrix multiplication is essential for modern deep learning architectures. It is used extensively in transformers and large language models. Major tech companies are greatly increasing model sizes and pushing towards increased computational requirements. Therefore, optimizing matrix multiplication becomes essential for both training and running inference, especially as GPUs and semiconductor manufacturing are becoming scarce and cannot keep up with demand. This paper discusses efficient matrix multiplication techniques and explores how quantum algorithms can lead to future computational improvements in AI and ML. We distinguish between verification algorithms and actual multiplication algorithms, and examine how quantum approaches, sparsity techniques, and algorithmic optimizations can lead to reduced computation in LLMs.**

***Index Terms*—Matrix multiplication, quantum algorithms, transformers, large language models, computational efficiency**

I. Introduction

Matrix multiplication is the backbone of almost all AI and neural network operations. For example, feedforward layers and attention mechanisms in transformers all rely on extensive matrix multiplications. In large language models, matrices are used to represent embeddings, weights, and activations. These are repeatedly multiplied, leading to substantial computational costs and time for training and inference. As the latest frontier models constantly increase to billions of parameters, we need to find ways to reduce the time complexity and cost of matrix operations, as it is becoming super expensive.

Originally, quantum matrix multiplication algorithms have provided improvements over the naive $O(n^3)$ approach. Strassen's algorithm gives an improvement to $O(n^{2.81})$, and the Coppersmith Winograd algorithm provides an improvement to $O(n^{2.376})$. These are still significant improvements compared to the classical $O(n^3)$ method. However, these classical improvements have practical limitations due to large constant factors and numerical stability issues.

II. Verification vs. Multiplication

It is crucial to distinguish between two different types of matrix algorithms. there are those that perform matrix multiplication and those that verify matrix multiplication has been done correctly.

Matrix Multiplication Algorithms actually compute the product $C = A \times B$ from scratch. Examples:

- Naive algorithm (Classical): $O(n^3)$
- Strassen's algorithm (Classical): $O(n^{2.81})$
- Coppersmith Winograd (Quantum): $O(n^{2.376})$

Matrix Verification Algorithms check whether a product $C = A \times B$ is correct, assuming someone has already computed C. They do not compute the multiplication themselves. Examples:

- Freivalds' algorithm (classical): $O(n^2)$
- Buhrman and Spalek (Quantum): $O(n^{5/3})$

The distinction is important because verification algorithms are useful for distributed computing scenarios is to check if a product is computed correctly, or for error detection in hardware. However, they do not directly help us compute matrix products faster. In the context of LLMs and ML, we need algorithms that actually perform multiplication efficiently.

III. Related Work

We see that Freivalds' algorithm shows a way to verify matrix multiplication in $O(n^2)$ time using probabilistic methods. Similarly, Buhrman and Spalek introduced a quantum algorithm achieving $O(n^{5/3})$ verification by using techniques like quantum walks and amplitude amplification. This method is great at confirming that $A \times B = C$ without recomputing the entire product. In this approach, they use superposition to check multiple entries at once, giving us a theoretical speedup for verification tasks.

However, for actually computing matrix products in LLMs, we need different approaches. There is a lot of work on reducing computational redundancy through quantum optimization and parallel efforts in deep learning. For example, techniques like sparse matrix multiplication, low rank approximation, and quantization are used to reduce unnecessary operations while maintaining performance. We know transformers rely heavily on quadratic attention operations, so quantum versions could provide computational benefits at scale.

IV. Quantum Algorithms for Matrix Multiplication

There have been Recent developments in quantum computing offer promising approaches for actually computing matrix products.

Quantum linear algebra techniques can potentially accelerate matrix operations by using the following techniques:

- Block encoding. This means representing matrices as quantum states.
- Quantum Singular Value Transformation (QSVT) for Applying matrix functions
- Quantum amplitude amplification for enhancing computational probabilities

These methods can provide polynomial or exponential speedups for certain matrix operations under specific conditions like when dealing with sparse matrices or when approximate solutions are acceptable.

For quantum algorithms that actually multiply matrices, the complexity depends on the following factors: matrix norms (condition numbers), sparsity of the matrices, required precision (ϵ), and encoding overhead. The runtime is typically shown as $O(\text{poly}(\log n, \kappa, 1/\epsilon))$ where κ is the condition number, compared to classical $O(n^3)$ or $O(n^{2.81})$ algorithms.

V. Quantum Transformer Inference Speedup

Recently there have been publications that have shown quantum algorithms for accelerating transformer inference using quantum linear algebra techniques. In this section,lets see exactly how quantum algorithms can speed up each part of transformer inference with concrete examples.

To understand the quantum speedup, we first need to understand what computations transformers actually do. When a transformer processes an input sequence X with n tokens and dimension d, it does lots of matrix multiplications in each layer. First, it creates Query, Key, and Value matrices through $Q = XW_Q$, $K = XW_K$, and $V = XW_V$. These three matrix multiplications each have $O(nd^2)$. Then it computes attention scores by multiplying QK^T, which costs $O(n^2d)$ operations. After applying softmax, it computes the attention output $O = AV$ for other $O(n^2d)$ operations. Then the feedforward network performs two more large matrix multiplications. In larger models, just one transformer layer requires billions of operations and a single forward pass needs tens of trillions of operations. This is why inference is so expensive and slow.

The key idea behind quantum matrix multiplication is block encoding. We can represent a classical matrix A as part of a quantum unitary operator. For a weight matrix W_Q in the QKV projection, we can create a quantum state where the matrix is encoded with a normalization factor $\alpha_w = ||W_Q||$. The encoding process takes $O(\log d)$ qubits instead of storing all d^2 entries. This is even better in sparse matrices because we only need to encode the non zero entries. This encoding overhead is important because if it takes too long to prepare the quantum state we lose the speedup advantage.

We can speed up computing $Q = XW_Q$ using quantum algorithms. Using classical methods, this takes $O(nd^2)$ operations because we need to multiply an $n \times d$ matrix by a $d \times d$ matrix. Using quantum algorithms, we can do this faster by using quantum singular value transformation. We encode each input vector from X into a quantum state using amplitude encoding, which only needs $O(\log d)$ qubits per vector. Then we apply the block encoded weight matrix using QSVT techniques. The quantum complexity becomes $O(\alpha_w \kappa \log^2(d/\epsilon))$ where κ is the condition number and ϵ is how precise we want the answer. In weight matrices where κ is close to 1 and α_w is around $\sqrt{d}$, we get a complexity of roughly $O(\sqrt{d}\log^2 d)$ instead of $O(d^2)$. This is a polynomial speedup. This means we will go from about millions of operations down to around thousands of quantum operations per token.

The attention mechanism computes $S = QK^T/\sqrt{d}$. in classical methods this costs $O(n^2d)$ operations. This is expensive because it scales quadratically with sequence length. For $n = 2048$ and $d = 12288$, this is about 51 billion operations. Quantum algorithms can exploit the structure of this computation by encoding each row of Q and K as quantum states. Then we can compute the dot products $\langle q_i|k_j\rangle$ using quantum amplitude amplification. This is where we get a big speedup. Instead of needing $O(n)$ operations to compute each attention score we can use Grover based amplitude amplification to reduce this to $O(\sqrt{n})$. The total quantum complexity becomes $O(\tilde{\alpha}_s\sqrt{n}d\log(1/\epsilon))$ where $\tilde{\alpha}_s$ is the encoding factor for the attention matrices. For normalized attention where $\tilde{\alpha}_s = O(1)$, we get $O(\sqrt{n}d\log(1/\epsilon))$ instead of $O(n^2d)$. This is a quadratic speedup in sequence length. For our example with $n = 2048$, we reduce 51 billion operations down to around 1 million quantum operations. This speedup is even better for longer sequences.

After softmax, attention matrices become sparse. Most attention weights are very close to zero, and only a few entries per row are significant. In local attention patterns, each token only attend strongly to $O(\sqrt{n})$ nearby tokens instead of all n tokens. Quantum algorithms can use this sparsity when computing $O = AV$ by using quantum walk based algorithms that only need to look at the non zero structure of the attention matrix. If the sparsity is $s = O(\sqrt{n})$ non zero entries per row, the quantum complexity becomes $O(s\alpha_A\alpha_V \log(n/\epsilon))$ instead $O(n^2d)$ in classical methods. This means for sparse attention with $s = O(\sqrt{n})$, we get complexity around $O(\sqrt{n}\log n)$ per output element. T is much better than the classical quadratic scaling.

One interesting thing about quantum implementations is that row wise operations like softmax and layer normalization also become really efficient. These operations can be done with quantum circuits of depth $O(\log d)$, which is independent of the actual dimension d. With classical methods these take $O(d)$ operations per row. Non linear activations like ReLU and GELU can be implemented with quantum arithmetic circuits that have logarithmic depth instead of linear cost. This means the total cost for all element wise operations becomes $O(n \log d)$ instead of $O(nd)$.

Let's calculate the total speedup for a complete transformer layer. For $n = 2048$ tokens and $d = 12288$ dimensions, the classical cost is roughly 3×10^{12} operations per layer when we add up the QKV projections at about 9×10^{11} operations, attention computation at about 5×10^{10} operations, and the feedforward network at about 2.5×10^{12} operations. With quantum algorithms and reasonable assumptions where $\alpha = O(\sqrt{d})$, sparsity $s = O(\sqrt{n})$, and precision $\epsilon = 10^{-3}$,

the quantum cost becomes about 10^8 quantum gates per layer. The QKV projections need around $\sqrt{12288} \times 2048 \times \log^2(12288) \approx 10^8$ quantum gates, attention needs about $\sqrt{2048} \times 12288 \times \log(2048) \approx 10^6$ quantum gates, and the FFN is similar to QKV . It needs around 10^8 quantum gates. This is roughly a 10000 times reduction in computational complexity. For a 96 layer model, this means we go from 3×10^{14} classical operations to just 10^{10} quantum operations.

The quantum speedup is best under certain conditions. We get the most benefit when the sequence length is long because that's where the quadratic attention cost really hurts. We also need weight matrices that are well conditioned with condition numbers less than 10. The attention patterns should also show natural sparsity, which they usually happpens after softmax. And we need to be okay with moderate precision around $\epsilon = 10^{-3}$, which is generally fine for inference.

The main practical challenge is the quantum state preparation overhead. We need to load classical data into quantum states, which currently takes $O(nd)$ time. This can cancel out the computational speedup we just calculated. However, recent proposals for Quantum Random Access Memory could reduce this loading time to $O(\log(nd))$, which would make the quantum advantage actually practical. Until we get efficient QRAM, hybrid approaches where we only use quantum acceleration for the most expensive operations are probably the most realistic path forward.

VI. Practical Considerations and Hardware Requirements

To implement quantum matrix multiplication algorithms, we need fault tolerant quantum computers with sufficient qubits and low gate error rates. There are three main methods for this:

Parameterized Quantum Circuits (PQC) methods are suitable for Noisy Intermediate Scale Quantum (NISQ) devices. These work with quantum state vectors to simulate transformer components with learnable parameters. These transformers are hardware efficient but are limited by qubit numbers, circuit depths, and training challenges.

Quantum Linear Algebra (QLA) Based Methods use algorithms like block encoding and quantum singular value transformations to speedup attention matrix calculations and linear transformations. These require fault tolerant quantum computers but offer great theoretical speedups.

Hybrid Classical Quantum Architectures are the most achievable strategy in the near term. In this approach, classical GPUs manage the machine learning pipeline while quantum computers accelerate specific intensive matrix operations. The Quantum Random Access Memory (QRAM) model allows for efficient vector storage and parallelization, achieving quadratic speedups for operations like leverage score sampling and approximate least squares calculations.

VII. Classical Optimization Techniques for LLMs

As quantum computers get better, classical optimization are still important for reducing computational costs in LLMs.

Transformers naturally show sparsity in attention patterns. We can take advantage of this sparsity through specialized algorithms and hardware to reduce computational requirements. We can also use techniques include sparse attention patterns (local, strided, block sparse), pruning redundant connections, and dynamic sparsity during training.

In low rank approximations, a lot of weight matrices in neural networks can be approximated by lower rank matrices without significant performance loss. This uses methods like Singular Value Decomposition (SVD), low rank factorization of weight matrices, and tensor decomposition for multi dimensional parameters.

In quantization, reducing precision of matrix elements from 32 bit to 8 bit or lower can reduce memory bandwidth and computational requirements through post training quantization, quantization aware training, and mixed precision training.

VIII. Integration with Classical Optimization Techniques

The most practical approach for the foreseeable future is using Hybrid algorithms which involves integrating quantum algorithms for actual computation with classical optimization techniques.

In Transformers Quantum algorithms can exploit sparsity through specialized block encoding, reducing gate counts for sparse matrices. This is particularly relevant for sparse attention matrices in transformers.

In Quantum Enhanced Low Rank Approximation, Quantum singular value estimation techniques can be combined with classical low rank approximation methods. we can adjust error tolerances to match quantization noise, and quantum amplitude encoding will naturally align with quantization techniques.

We can also do gradient descent computation, mainly during backpropagation in neural network training. we can reduce redundancy in gradient computation by using quantum accelerated sampling and approximation methods. This can speed up validation of partial results from attention block updates in LLMs without loss of accuracy.

IX. Impact on LLMs and Transformers

Transformers heavily use self attention blocks, which involve multiple matrix multiplications per layer. For example, the computations QK^T, $(QK^T)V$, and feedforward transformations are major components of transformer costs and runtime. These show quadratic scaling with input sequence length, making training and inference expensive and time consuming.

Quantum algorithms for actual matrix multiplication could potentially reduce the asymptotic complexity of attention computations, accelerate gradient calculations during training, enable faster inference for production systems, and allow for larger context windows in language models. Combined with classical optimizations like sparsity and quantization, quantum approaches could dramatically improve the scalability of future LLMs.

X. Specific Use Cases and Applications

During training, matrix multiplications make up a large portion of the computational budget. Quantum acceleration could reduce training time from months to weeks, enable model training with trillions of parameters, and lower energy consumption and environmental impact.

This could also help with the actual real time inference. In production systems serving millions of users, quantum accelerated inference could reduce latency for real time applications, decrease serving costs, and enable more complex models in resource constrained environments.

Quantum matrix could also improve scientific Computing and other AI Applications more than just LLMs, quantum matrix multiplication could also accelerate molecular dynamics simulations, climate modeling, drug discovery using AI, and financial modeling and risk analysis.

XI. Future Directions

Future work should explore hybrid algorithms that use both classical and quantum computation within the same pipeline. For example, quantum algorithms could handle specific matrix operations while classical systems manage data preprocessing and post processing. As quantum hardware improves, integration of quantum algorithms for actual multiplication (not verification) could greatly speed up AI training and inference pipelines.

Furthermore, classical simulations of quantum inspired algorithms offer improvements for GPU based architectures in sparse or structured matrix contexts. Research where algorithms and hardware are developed together will also be essential because currently we are trying to make make quantum algorithms that will fit into classical algorithms. but it might be more beneficial to built quantum first AI that take full advantage of quantum hardware.

Areas requiring further investigation include optimal partitioning of computations between classical and quantum systems, development of quantum native neural network architectures, error mitigation strategies for NISQ devices, scalable QRAM implementations, and quantum algorithms for specific matrix structures common in transformers.

XII. Limitations and Challenges

Eventhough there are promising theoretical speedups, several issues must be overcome before quantum matrix multiplication has widespread adoption.

One of this is the quantum state preparation overhead. The cost of encoding classical data into quantum states can offset computational gains form using the quantum algorithms. so we need efficient encoding schemes are essential for a practical advantage.

Another major limitation is The hardware limitations. current quantum hardware suffers from limited qubit counts generally $<$1000 qubits, high error rates which need a lot of error correction, short coherence times limiting computation depth, and limited connectivity between qubits.

There are also some algorithmic specific constraints in this area. Quantum advantages are often dependent on matrix structure (sparsity, condition number), required precision, and problem size (may need very large matrices for advantage).

Another issue is architecture issues. Current neural network architectures are designed for classical computers. To fully exploit quantum advantages, we need to design quantum first model architectures from first principles, rather than trying to accelerate individual components of classically optimized models using quantum algorithms.

XIII. Conclusion

This paper discusses efficient matrix multiplication techniques essential for large language models and transformer architectures. While classical algorithms provide incremental improvements over naive methods, quantum approaches for actual matrix multiplication, such as quantum linear algebra techniques and recent quantum transformer implementations, provide theoretical speedups through superposition and amplitude amplification.

To get these quantum based improvements, we need fault tolerant hardware, hybrid architectures that uses both quantum and classical computation, and quantum native algorithm design. Integration with classical optimization to create hybrid algorithms with techniques such as sparsity exploitation, low rank approximation, and quantization show the most potential for near term progress in this area. The future of large language models may depend on successfully combining quantum acceleration with classical optimizations to overcome current computational bottlenecks.

References

[1] H. Buhrman and R. Spalek, "Quantum Verification of Matrix Products," arXiv:quant ph/0409035v2, 2005. Available: https://arxiv.org/abs/quant ph/0409035v2

[2] A. Vaswani et al., "Attention Is All You Need," in *Advances in Neural Information Processing Systems*, 2017.

[3] D. Coppersmith and S. Winograd, "Matrix Multiplication via Arithmetic Progressions," *Journal of Symbolic Computation*, vol. 9, 1990.

[4] R. Freivalds, "Fast Probabilistic Algorithms," in *Proceedings of the 8th Symposium on Mathematical Foundations of Computer Science*, Springer, 1979.

[5] V. Strassen, "Gaussian Elimination Is Not Optimal," *Numerische Mathematik*, vol. 13, 1969.

[6] N. Guo et al., "Quantum Transformer: Accelerating Model Inference via Quantum Linear Algebra," arXiv:2402.16714v3, 2025.

[7] I. Kerenidis and A. Prakash, "Quantum Algorithms for Linear Algebra and Machine Learning," Ph.D. Dissertation, University of California, Berkeley, 2014.

[8] Y. Tang et al., "Quantum Machine Learning: A Review of Recent Progress and Challenges," *IEEE Transactions on Neural Networks and Learning Systems*, 2024.

[9] S. Lloyd et al., "Quantum Algorithms for Linear Systems of Equations," *Physical Review Letters*, vol. 103, 2009.