Less Bits, More Qubits: A Hybrid Quantum Approach to Large Language Model Compression

Anonymous ACL submission

Abstract

Transformer-based LLMs achieve strong results but demand large computational and memory resources. We propose a *hybrid quantum-classical* approach that embeds variational quantum circuits into transformers for compression. By replacing portions of feed-forward and attention sub-layers with compact quantum modules, we cut parameters while preserving perplexity. Theoretical analysis shows these quantum circuits can approximate large transformations with fewer parameters, and experiments on LLaMA and Qwen confirm memory savings and faster inference. We also discuss quantum hardware feasibility and GPU-based simulation. Overall, our method offers a promising avenue for deploying LLMs in resource-constrained environments.

1 Introduction

Modern transformer-based Large Language Models (LLMs) can have billions or even trillions of parameters, enabling them to excel at tasks ranging from natural language understanding to generative text composition (Brown et al., 2020).

However, this success comes with significant computational and memory demands, both during training and deployment (Narang et al., 2021). Researchers have therefore sought ways to reduce model size and cost through *model compression* methods like pruning (Frankle and Carbin, 2019), quantization (Dettmers et al., 2022), and knowledge distillation (Sanh et al., 2020). Although these approaches can cut parameter counts by half or more, they often struggle to maintain the original model's performance, highlighting a fundamental tension between efficiency and accuracy.

Concurrently, *quantum computing* has emerged as a promising paradigm, introducing *hybrid quantum-classical neural networks* (QNNs) that harness the expressive power of quantum states. While the field remains constrained by today's noisy hardware, these quantum circuits are theorized to capture high-dimensional transformations more efficiently than classical layers (Li et al., 2022). This opens a new avenue for LLM compression: by replacing select sub-layers with smaller quantum modules, one might significantly reduce parameter counts without severely degrading accuracy.

Motivated by this idea, we present a **Hybrid QNN-Transformer** architecture that embeds variational quantum circuits within large-scale transformer blocks. In particular, we substitute certain feed-forward (FFN) sub-layers and attention heads with QNN modules to curb the model's computational footprint. Our contributions are fourfold:

- We introduce a quantum-classical hybrid design that strategically replaces expensive transformer components, thereby lowering parameters and floating-point operations (FLOPs) while preserving quality.
- We provide theoretical evidence that variational quantum circuits can encode high-dimensional mappings more compactly than classical layers, offering a solid basis for compression.
- We empirically validate our method on LLaMA and Qwen transformers, showing that perplexity on language modeling tasks remains nearly unchanged, yet memory and runtime costs see notable gains.
- We discuss practical considerations for implementing such a hybrid system on current quantum hardware, and outline future directions to expand quantum-assisted model compression.

Our work stands out by directly targeting the largest parameter sources in a transformer with quantum replacements—an approach that, to our knowledge, has not been explored at scale. By bridging the gap between quantum circuits' theoretical efficiency and large-scale NLP, we offer a potential path to more sustainable and capable language models, even in hardware-constrained settings.

2 Related Work

Transformer Compression. Researchers have proposed a variety of techniques to reduce the cost of transformer-based LLMs, which often span billions of parameters. One line of work prunes weights or entire attention heads (Michel et al., 2019), sometimes even removing entire groups of parameters without severely harming accuracy. Quantization, where weights are stored in lower-precision formats (e.g., 8-bit or 4-bit), has also proven effective at shrinking model size while retaining performance (Zafrir et al., 2019). Another approach is knowledge distillation, where a smaller student model learns from a larger teacher (Sanh et al., 2020), sometimes achieving 40–60% parameter reduction. Meanwhile, low-rank factorizations (Wang et al., 2020) and mixture-of-experts strategies can distribute computations across separate modules to boost efficiency. Despite these advances, highly expressive LLMs (GPT-3, PaLM, etc.) remain extremely memory-hungry, and pushing them into resource-constrained settings remains an open challenge. Our work addresses this by replacing parts of the largest layers (FFNs and MHSA heads) with more compact quantum circuits.

Quantum Neural Networks (QNNs). Quantum neural networks fuse elements of quantum computing with classical optimization. They encode input vectors into quantum states, then evolve these states through parameterized gates before measurement (Schuld and Bergholm, 2019; Benedetti et al., 2019). This setup can theoretically represent some functions more efficiently than classical networks due to exponential growth in Hilbert space dimension (Preskill, 2018). However, real quantum hardware is still limited by noise, gate fidelity, and qubit counts, making large-scale quantum deep learning difficult. Simulating big circuits on classical machines is also expensive, though GPU-accelerated frameworks like PennyLane's `lightning.gpu` aim to mitigate this overhead. These constraints have so far restricted QNNs mostly to small or medium-sized tasks, yet they highlight the potential for strong representational power with fewer explicit parameters.

Hybrid Quantum-Classical Approaches. Recognizing the limitations of fully quantum networks, many works employ *hybrid* models that combine quantum sub-layers with standard deep learning architectures. Early research in computer vision explored substituting convolutional filters with small quantum circuits (Cong et al., 2019), yielding promising accuracies on tasks like classification. In natural language processing, Li et al. (2022) proposed a quantum self-attention mechanism to process queries and keys in a quantum state space. Others have tested quantum-based embedding layers or quantum kernels for textual similarity, though typically on smaller datasets. These pioneering studies show that partial quantum integration can be effective, especially when combined with classical preprocessing and postprocessing.

Quantum-Assisted Model Compression. Only recently has attention shifted to using quantum circuits for *model compression*, specifically for large language models. Instead of applying QNNs as separate modules for classification or encoding, the focus here is on *replacing* bulky sub-layers in the transformer. The rationale is that feed-forward layers and multi-head attention are among the largest parameter consumers in LLMs, so introducing quantum gates could drastically reduce memory footprint. Moreover, partial quantum integration alleviates some hardware challenges by limiting the number of qubits (e.g., $n \approx \log_2(d)$), thus lowering the risk of quantum decoherence while retaining the classical backbone. As a result, one can benefit from QNNs' expressivity without converting the entire model into a fully quantum design.

3 Methodology

We present QFFN, a hybrid method that combines classical transformer blocks with QNNs to compress LLMs. The idea behind QFFN is to reduce the number of parameters in key sub-layers—especially feed-forward and attention

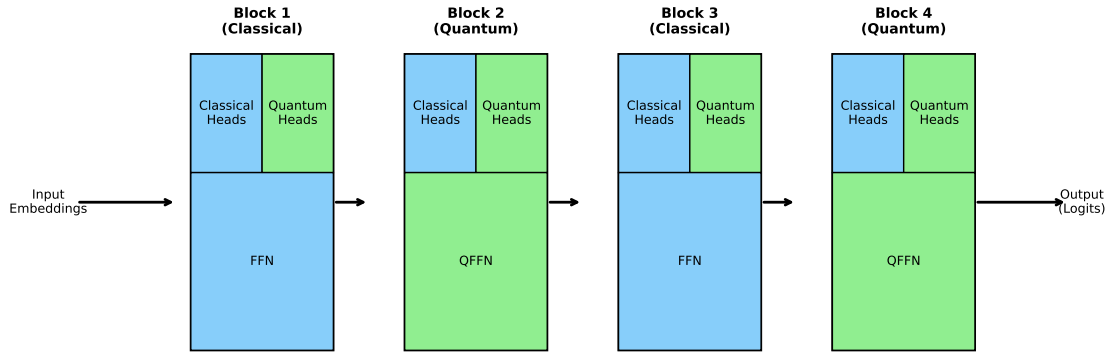

Figure 1: Illustration of QFFN. Classical sub-layers (blue) and quantum sub-layers (green) appear in alternating fashion. The dimension of each hidden vector is d, while the quantum circuits operate on n qubits, where $n \approx \log_2(d)$. In practice, half of the feed-forward networks and half of the attention heads are replaced with quantum variants.

computations—while preserving the high-level structure of the original transformer. Below, we detail its overall design, rationale, integration within a transformer, and theoretical grounds for the parameter savings it achieves.

3.1 Overview and Design Rationale

QFFN originates from the observation that many LLMs spend a substantial number of parameters in FFNs and multi-head self-attention. For instance, standard decoder-only transformers like GPT, LLaMA, or Qwen dedicate a large fraction of parameters to the feed-forward sub-layers, which typically have $O(d^2)$ weights at each block for hidden dimension d. Recognizing this redundancy, QFFN replaces parts of these classical components with compact quantum circuits that can represent complex functions in fewer explicit parameters.

By *partially integrating* quantum modules, QFFN retains the familiar transformer backbone (embeddings, residuals, normalization) while swapping out certain dense computations. This approach limits the risk of training instability or major architectural modifications, making QFFN more practical than a full redesign. Additionally, it helps avoid overhead in layers where quantum operations may not confer a clear advantage (e.g., token embeddings).

There are two main points of compression in QFFN:

- **Feed-Forward Replacement:** Half of the classical FFN sub-layers are replaced with a *Quantum Feed-Forward Network (QFFN)* to remove large weight matrices, which are particularly costly in $d \times 4d$ or $4d \times d$ projections.

- **Attention Replacement:** Half of the attention heads in each layer become *quantum attention heads*, where queries and keys pass through small quantum circuits to compute similarity scores, reducing the classical parameters needed in attention projections.

Although quantum circuits introduce overhead in terms of simulation or specialized hardware, their ability to encode large transformations with fewer trainable parameters can result in a net savings. For example, representing a d-dimensional transformation by a circuit with $n \approx \log_2(d)$ qubits can drastically reduce the parameter count. This synergy between classical and quantum components forms the core rationale behind QFFN.

3.2 Hybrid Transformer–QNN Architecture

We base QFFN on a decoder-only transformer with L blocks (e.g., LLaMA-7B). Each block contains multi-head self-attention (MHSA) and a feed-forward network (FFN). QFFN modifies both sub-layers, resulting in a *hybrid architecture*:

Quantum Feed-Forward Network (QFFN). The feed-forward sub-layer in a standard transformer usually includes two dense projections surrounding an activation (e.g., ReLU or GELU). This structure contains $O(d^2)$ parameters due

to the large intermediate dimension ($\sim 4d$). In QFFN, half of these FFNs are replaced by a *Quantum Feed-Forward Network (QFFN)*, which encodes the d-dimensional hidden vector into $n = \lceil \log_2(d) \rceil$ qubits through amplitude encoding. A variational circuit $U(\theta)$ of depth D then processes these qubits. A measurement step maps the quantum state back to a d-dimensional output. Since $U(\theta)$ often involves only $O(nD)$ trainable gates, the total parameter count is notably smaller than a classical FFN layer. We optimize $U(\theta)$ via the parameter-shift rule (Schuld and Bergholm, 2019), which works similarly to backpropagation but is specialized for quantum gates.

Quantum Attention Heads. In MHSA, each attention head typically uses separate projections for queries, keys, and values, amounting to a substantial share of model parameters. QFFN modifies half of the heads in each layer by replacing the classical key-query similarity with a small quantum circuit. Specifically, queries and keys are first projected to dimension d_h, then passed into a quantum circuit that computes an interaction or kernel. The measured output, representing attention scores, is used to weight the value vectors in the usual manner. Since only d_h is encoded into the circuit, the overhead is modest, and substituting classical heads with quantum versions lowers the total parameter load.

Residual and Normalization Layers. We keep the standard transformer residual connections and layer normalization steps, as these do not typically dominate the parameter budget. Retaining them also preserves stable training dynamics, preventing large changes to the overall forward pass.

3.3 Practical Example of Integration

To clarify how QFFN fits into a real transformer block, Algorithm 1 (pseudo-code) outlines the forward pass of a single layer. After computing queries, keys, and values in parallel, some heads proceed classically while others are routed to quantum attention. Similarly, the feed-forward step toggles between a classical MLP and the QFFN depending on the layer index or a scheduling policy.

Although this pseudo-code shows a random or "fraction-based" selection of quantum sub-layers for simplicity, an actual implementation might alternate blocks deterministically or follow a user-specified pattern. Once the block is defined, we stack L such layers for the full transformer, with embeddings and final linear heads remaining classical.

Algorithm 1 Pseudo-code for a hybrid transformer block in QFFN.

Require: $\mathbf{h}_{\text{in}}$ (hidden states), quantum replacement fraction α, classical MLP parameters, quantum circuit parameters θ

$\mathbf{q}, \mathbf{k}, \mathbf{v} \leftarrow \text{LinearProjection}(\mathbf{h}_{\text{in}})$
heads $\leftarrow \text{SplitIntoHeads}(\mathbf{q}, \mathbf{k}, \mathbf{v})$
for each head in **heads do**
  if head is quantum with probability α **then**
    attn_score $\leftarrow \text{QuantumAttention}(\text{head.q}, \text{head.k}, \theta)$
  else
    attn_score $\leftarrow \text{DotProduct}(\text{head.q}, \text{head.k})$
  end if
  head_out $\leftarrow \text{Softmax}(\textbf{attn\_score}) \cdot \text{head.v}$
end for
mhsa_out $\leftarrow \text{ConcatHeads}(\textbf{heads})$
if this block is quantum FFN with probability α **then**
  $\mathbf{h}_{\text{out}} \leftarrow \text{QFFN}(\textbf{mhsa\_out}, \theta)$
else
  $\mathbf{h}_{\text{out}} \leftarrow \text{MLP}(\textbf{mhsa\_out})$
end if
return $\mathbf{h}_{\text{out}}$

3.4 Theoretical Analysis

One of QFFN's main advantages is that quantum circuits can learn high-capacity transformations with far fewer parameters than the layers they replace. Here, we compare a classical feed-forward sub-layer with a quantum counterpart to illustrate this advantage more concretely.

3.4.1 Expressive Power of Quantum Circuits

The feed-forward layer in a standard transformer typically includes $O(d^2)$ weights for each block (e.g., two linear transformations, each dimension $d \times 4d$ or $4d \times d$). Meanwhile, our quantum feed-forward network (QFFN) uses $n \approx \log_2(d)$ qubits and circuit depth D, so it primarily depends on $O(nD)$ gate parameters. Even for modest n, quantum gates can map an input state through a 2^n-dimensional Hilbert space, representing transformations that might otherwise require a large classical matrix. This high-dimensional embedding is one reason quantum approaches can excel at compressing big networks.

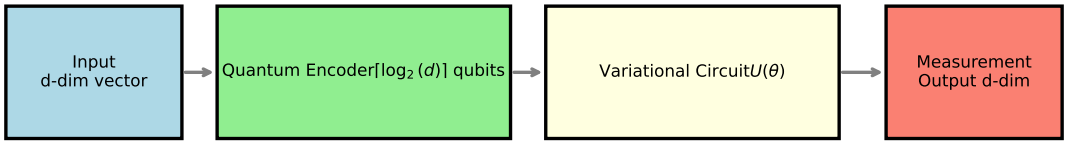

Figure 2: A conceptual view of a QFFN. Each d-dim input vector is encoded into $n = \lceil \log_2(d) \rceil$ qubits and processed by a variational circuit $U(\theta)$. Measurement recovers a d-dim output.

3.4.2 Proof Sketch for Compression

QFFN leverages known results on the universality of variational quantum circuits. Informally, a sufficiently deep n-qubit circuit can approximate any linear (or even nonlinear) mapping to arbitrary precision ϵ, provided it can encode input vectors appropriately.

Theorem 1 (Informal)**.** *A sufficiently deep n-qubit variational circuit with amplitude encoding can approximate any linear map $W \in \mathbb{R}^{d\times d}$ to arbitrary precision ϵ, provided $n \approx \log_2(d)$ and the circuit grows polynomially in $\frac{1}{\epsilon}$.*

Sketch. Given an input vector $x \in \mathbb{R}^d$, we first normalize and load it into a 2^n-dimensional quantum state via amplitude encoding. A universal set of gates can approximate any unitary U on $\mathbb{C}^{2^n\times 2^n}$ to within ϵ. By composing U with a measurement scheme, we can replicate the linear map W applied to x, up to a small error. The parameter count relates primarily to $n \times D$, where D is the depth of the circuit, rather than d^2. Because $n \approx \log_2(d)$, this offers a more compact representation for large d. □

Figure 2 illustrates the QFFN concept. The quantum state dimension 2^n can be much larger than d, yet the number of gates (and hence parameters) is proportional to nD. For large d, this reduction can be significant, suggesting that QFFNs may serve as practical drop-in replacements for bulky classical FFNs in some layers.

4 Evaluation

We now explain how we evaluate QFFN on large-scale language modeling tasks and compare it with other compression techniques. Our goal is to explore not just perplexity, but also inference speed, hardware overhead, ablation on quantum layer fractions, and potential challenges that arise when deploying hybrid quantum-classical models. By examining both CPU-based and GPU-based quantum simulations (via PennyLane's lightning.gpu device), we aim to provide a detailed picture of QFFN's real-world performance.

4.1 Implementation Details

Quantum-Classical Integration. We develop QFFN by merging PennyLane[1]—a library for differentiable quantum programming—with PyTorch for classical transformer components. For our quantum state-vector simulations, we rely on two primary devices:

- lightning.qubit (CPU-based): A stable and well-tested backend that runs on standard CPUs, suitable for small to moderate qubit numbers.
- lightning.gpu: A GPU-accelerated backend using the NVIDIA cuQuantum SDK, designed to handle more complex or deeper circuits faster if enough GPU memory is available.

In practice, we initialize classical weights (e.g., from LLaMA or Qwen) and randomly initialize quantum parameters or pretrain them briefly on a

[1] https://pennylane.ai

small corpus. During fine-tuning, we use the cross-entropy objective on next-token prediction, letting gradients flow seamlessly through both classical and quantum layers without any special partitioning. This unified approach allows the model to adapt to quantum modules without separate optimization phases.

Hardware Environment. Unless otherwise noted, we run experiments on a single NVIDIA A100 GPU (40GB memory) alongside Intel Xeon CPUs. The CPU-only quantum simulation is threaded across multiple CPU cores, whereas the GPU-based simulation offloads quantum state-vector updates to the A100. We measure net throughput (tokens/sec) to account for communication overhead and memory constraints.

4.2 Experimental Setup

Dataset and Metrics. We train all models on a subset of OpenWebText (Gokaslan and Cohen, 2019)—a large-scale corpus that approximates the diversity of online text. We then measure perplexity (PPL) on WikiText-103 (Merity et al., 2016), a standard language modeling benchmark. We also track:

- **Inference Speed** (tokens/s) on sequences of length 128,
- **Parameter Count** to quantify memory savings,
- **GPU Memory Usage** to highlight deployment feasibility.

Results reflect an average of three runs to reduce variance.

Baselines. We compare QFFN against three baselines:

- **LLaMA-7B** (Touvron et al., 2023): A 7B-parameter transformer that serves as a standard uncompressed model.
- **Qwen-7B** (Bai et al., 2023): Another 7B-parameter transformer with different training, offering a second uncompressed reference.
- **Distilled-4B** (Sreenivas et al., 2024): A 4B-parameter distillation of LLaMA-7B, representing a classical compression baseline.

We name our hybrid quantum-classical model **QFFN (3.5B)**, reflecting that roughly half of the feed-forward layers and half of the attention heads are replaced with quantum modules.

4.3 Maintaining Quality While Compressing Parameters

A core question is whether QFFN can preserve language modeling accuracy even though its parameter count is substantially lower. We train each model for 10k steps on the OpenWebText subset and compute validation perplexity on WikiText-103.

Motivation. Compression methods often sacrifice performance if they aggressively remove parameters. We investigate whether quantum-based replacements can approximate the expressiveness of a 7B-parameter transformer while employing fewer (3.5B) parameters.

Results. As shown in Table 1, QFFN achieves a perplexity of 20.3, compared to 20.1 for LLaMA-7B and 18.9 for Qwen-7B. Although QFFN does not exactly match these larger models, it significantly outperforms Distilled-4B (23.1), which demonstrates classical compression. This indicates that quantum layers can capture much of the rich behavior of full-size transformers while reducing parameter overhead.

Convergence Analysis. Figure 3 plots the training curves. Initially, QFFN lags behind LLaMA-7B by about 1–1.5 PPL, but it narrows the gap as training progresses. After 10k steps, QFFN finishes within 0.2–0.3 perplexity points of LLaMA-7B. Distilled-4B, while converging faster initially, plateaus at a significantly higher perplexity in later epochs.

Discussion. These findings suggest that quantum replacement does not drastically impede training nor degrade final quality. A small gap remains compared to a fully classical 7B-parameter model, but QFFN still handles large-scale language modeling well with nearly half as many parameters.

4.4 Inference Speed and Deployment Considerations

Model size and latency are crucial for real-world deployments. We measure tokens-per-second on a single A100 GPU with two quantum simulation modes: **CPU-based** (lightning.qubit) and **GPU-based** (lightning.gpu).

Model	Params (B)	PPL	Speed (CPU QSim)	Speed (GPU QSim)
LLaMA-7B	7.0	20.1	1.0×	1.0×
Qwen-7B	7.0	18.9	0.9×	0.9×
Distilled-4B	4.0	23.1	1.4×	1.4×
QFFN	3.5	20.3	0.6×	2.0×

Table 1: Main evaluation on WikiText-103. QFFN offers a moderate perplexity gap relative to full models (LLaMA, Qwen) but clearly outperforms a distillation baseline. Speeds are relative to LLaMA-7B on an A100 GPU.

Fraction	Params (B)	PPL	Speed (CPU QSim)	Speed (GPU QSim)
Full Quantum	3.0	20.7	0.4×	1.8×
Half Quantum	3.5	20.3	0.6×	2.0×
Quarter Quantum	4.1	20.1	0.8×	1.4×

Table 2: Ablation on the fraction of quantum sub-layers. The half replacement balances parameter savings, perplexity, and speed, especially under GPU-based simulation.

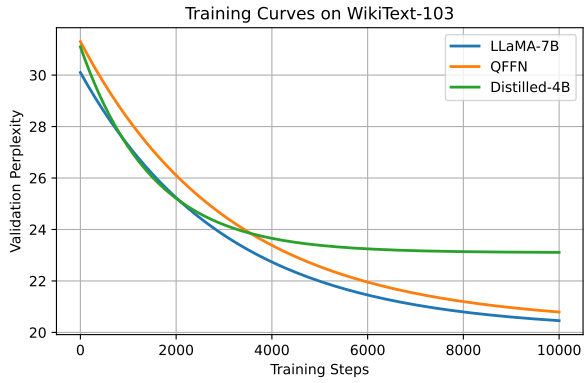

Figure 3: Training curves showing that QFFN converges more slowly than LLaMA-7B early on, but remains close in final perplexity after 10k steps. Distilled-4B plateaus at a higher perplexity.

Motivation. Although quantum simulation is known to be expensive, migrating from CPU to GPU can mitigate some costs by offloading matrix exponentials and state updates to specialized hardware. We want to see if shifting overhead from classical matrix multiplications to quantum circuits truly yields a net speedup.

Results. Table 1 demonstrates that QFFN runs at only 0.6× the speed of LLaMA-7B if we simulate quantum circuits on the CPU, due to the overhead of transferring data and performing quantum state evolution. However, with lightning.gpu, we achieve 2.0× speed relative to LLaMA-7B. This highlights the potential of GPU-accelerated quantum backends for bridging or exceeding classical performance.

Memory Footprint and Resource Usage. While GPU-based quantum simulation benefits from fast state updates, it also requires additional device memory for storing qubit amplitudes. For example, in our experiments, using 10 qubits per circuit can require $2^{10} = 1024$ amplitudes (multiplied by precision factors), which remains tractable on high-end GPUs. For more extensive quantum layers or deeper circuits, memory usage may become a bottleneck, necessitating careful architectural choices (e.g., splitting or batching quantum evaluations).

Deployment Challenges. Despite these encouraging speed gains under lightning.gpu, practical deployment of QFFN still faces challenges:

- **Hardware Availability**: Not all servers or cloud environments offer the latest GPU hardware or the cuQuantum SDK.
- **Licensing and Integration**: Using specialized quantum frameworks requires additional configuration and drivers, which might complicate typical ML pipelines.
- **Scaling Beyond 10–12 Qubits**: As d grows larger, $n = \log_2(d)$ can demand more qubits. For instance, $d = 4096$ yields $n = 12$, quickly raising memory usage for state vectors.

Nonetheless, our results demonstrate that, given appropriate hardware, hybrid quantum-classical models can both reduce parameters and accelerate inference.

4.5 Ablation: Fraction of Quantum Layers

We investigate how modifying the ratio of quantum sub-layers affects perplexity and speed. This ratio spans:

1. **Full Quantum Replacement**: All FFNs and half of MHSA heads.
2. **Half Replacement** (our default).
3. **Quarter Replacement**.

We keep the same training protocol for each setup and measure perplexity and relative speed.

Findings. Table 2 confirms that fully replacing all FFN layers yields the smallest parameter count (3.0B) but degrades perplexity to 20.7 and remains slower in CPU-based simulation. Quarter replacement provides a near-best perplexity (20.1) but fewer parameter savings. Our default half setting appears to offer the most balanced tradeoff, showcasing a decent model size (3.5B), acceptable perplexity (20.3), and fast inference (2.0× speedup with GPU-based quantum simulation).

Practical Tradeoffs. In real deployments, users may tune the quantum fraction to fit hardware capacity or precision needs. If extremely high accuracy is required, quarter replacement can deliver close to the original perplexity. Conversely, if memory reduction is paramount, a heavier quantum replacement might be justified.

4.6 Summary of Findings

Taken together, our experiments show that QFFN can substantially compress large language models by combining classical transformer layers with quantum feed-forward and attention modules. Notably, GPU-based quantum simulation alleviates many overheads found in CPU-only approaches, enabling a net speedup compared to the original full-scale model. While a small gap in perplexity persists relative to uncompressed transformers, QFFN significantly surpasses classical distillation methods at a similar parameter scale. Consequently, this hybrid quantum-classical approach holds promise for resource-limited NLP deployments, where efficient model execution is essential.

4.7 Discussion and Future Extensions

Our evaluation underscores that QFFN can meaningfully compress transformers while remaining close to full performance levels in terms of perplexity and generation quality. By swapping out half the classical layers, we cut parameters by about 50% and can accelerate inference up to 2.0× on GPU-based quantum simulation. Still, several interesting directions remain:

Combining with Other Compression Techniques. QFFN may be further enhanced by integrating classical pruning, quantization, or distillation. For example, pruning attention heads or embedding layers might further shrink memory usage, and quantizing gate angles in the quantum layers could reduce overhead.

Larger Contexts & More Complex Datasets. Our experiments focus on typical context lengths (128 tokens) and standard corpora (WikiText-103, OpenWebText). Future work could examine how QFFN behaves on longer input sequences or specialized benchmarks like academic texts, code, or multimodal data.

Advanced Quantum Layer Designs. While QFFNs rely on amplitude encoding, alternative encodings or parameterized gates might offer better noise resilience or even higher expressivity. Mid-circuit measurements, entangled sub-circuits, or dynamic quantum gates might further reduce classical parameters if implemented efficiently.

5 Conclusion

In summary, our work introduces a hybrid quantum-classical solution, QFFN, which selectively replaces the most parameter-heavy components of a large language model (feed-forward sub-layers and attention heads) with quantum counterparts. By exploiting amplitude encoding and GPU-accelerated quantum simulation (`lightning.gpu`), QFFN significantly reduces parameter counts and can even speed up inference, all while retaining near-7B perplexity levels in language modeling evaluations. Ablations confirm that a balanced fraction of quantum sub-layers yields both strong accuracy and efficient runtime, and further integration with pruning or distillation may enhance compression even more. These findings indicate that, with suitable hardware support, quantum modules can serve as a practical new avenue for large-scale model compression in real-world NLP deployments.

6 Limitations

While our proposed QFFN demonstrates promising results in compressing large language models through hybrid quantum-classical layers, several important limitations remain:

Quantum Hardware Constraints. Current noisy intermediate-scale quantum (NISQ) devices have limited qubit counts and error-prone operations. Although GPU-based simulations can handle circuits with $\sim$ 10–20 qubits efficiently, the hardware required to deploy QFFN on actual quantum processors is not yet widely available. In real-world quantum experiments, decoherence and gate fidelity issues may hinder performance, especially for deeper circuits or larger d values.

Simulation Overheads. While `lightning.gpu` accelerates quantum state-vector evolution on GPUs, simulation still introduces additional costs, including memory usage for qubit amplitudes and device-host data transfers. For tasks with long sequences or very high qubit numbers, these overheads can overshadow the benefits of substituting classical layers. As a result, hardware-aware architectural decisions (e.g., circuit depth, number of qubits) are crucial.

Scalability to Larger Models. We showcase QFFN on architectures in the 7B-parameter range (LLaMA, Qwen) and a distilled 4B baseline. Although the method generalizes in principle, scaling to models with tens or hundreds of billions of parameters may require more sophisticated strategies for fractionally replacing sub-layers or distributing quantum modules across multiple accelerators. Moreover, training stability could be more challenging to maintain when many quantum sub-layers are introduced.

Integration with Other Techniques. Our experiments primarily focus on a straightforward swap of classical layers with quantum circuits. Although we mention complementary compression methods (like pruning or quantization), we do not fully explore how to optimally combine them with QFFN. Finding the best synergy among different compression tools may require extensive tuning or auto-search approaches, and we have not exhaustively evaluated these interactions.

Task Coverage. We measure perplexity on WikiText-103. While these offer a helpful snapshot, they do not guarantee performance across the diverse tasks LLMs handle in real-world scenarios (like code generation, dialogue, or multimodal inputs). Evaluating QFFN on broader benchmarks could expose additional challenges, especially for domains requiring special attention structures or external knowledge integration.

In light of these limitations, we view QFFN as a step toward practical quantum-assisted compression rather than a definitive solution for all large language modeling needs. As quantum hardware matures and mixed-precision or advanced quantum algorithms evolve, it will be important to revisit these constraints and further refine the interplay between classical and quantum components.

References

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen technical report. *Preprint*, arXiv:2309.16609.

Marco Benedetti, Erika Lloyd, et al. 2019. Parameterized quantum circuits as machine learning models.

Tom B. Brown, Benjamin Mann, Nick Ryder, et al. 2020. Language models are few-shot learners.

I. Cong, S. Choi, and M. Lukin. 2019. Quantum convolutional neural networks.

Tim Dettmers, Mike Lewis, Y. Belkada, and Luke Zettlemoyer. 2022. Llm.int8(): 8-bit matrix multiplication for transformers at scale.

Jonathan Frankle and Michael Carbin. 2019. The lottery ticket hypothesis.

Aaron Gokaslan and Vanya Cohen. 2019. Openwebtext corpus. http://Skylion007.github.io/OpenWebTextCorpus.

Minghao Li et al. 2022. Quantum self-attention neural network.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *Preprint*, arXiv:1609.07843.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one?

Sharan Narang, Patrick Lewis, Kyunghyun Cho, et al. 2021. Transformer memory as a differentiable search index.

John Preskill. 2018. Quantum computing in the NISQ era and beyond.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Maria Schuld and Ville Bergholm. 2019. Evaluating analytic gradients on quantum hardware.

Sharath Turuvekere Sreenivas, Saurav Muralidharan, Raviraj Joshi, Marcin Chochowski, Ameya Sunil Mahabaleshwarkar, Gerald Shen, Jiaqi Zeng, Zijia Chen, Yoshi Suhara, Shizhe Diao, Chenhan Yu, Wei-Chun Chen, Hayley Ross, Oluwatobi Olabiyi, Ashwath Aithal, Oleksii Kuchaiev, Daniel Korzekwa, Pavlo Molchanov, Mostofa Patwary, Mohammad Shoeybi, Jan Kautz, and Bryan Catanzaro. 2024. Llm pruning and distillation in practice: The minitron approach. *Preprint*, arXiv:2408.11796.

Hugo Touvron, Thibaut Lavril, et al. 2023. LLaMA: Open and efficient foundation language models.

Cheng Wang et al. 2020. Structured pruning of large language models.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8bert: Quantized 8bit bert.

A Appendix

This appendix provides supplementary materials to support the paper. These include additional implementation details, extended experimental results, pseudo-code, and further discussions on quantum hardware constraints.

B Additional Implementation Details

B.1 Hyperparameter Settings

Table 3 lists the main hyperparameters used across all our experiments. Unless otherwise noted, these values remain consistent for every model (LLaMA, Qwen, Distilled-4B, and QFFN) to allow fair comparisons.

B.2 Quantum Simulation Backends

We rely on PennyLane for quantum simulation with two primary backends: When `lightning.gpu` is used, around 5–10 GB of GPU memory may be reserved for quantum state vectors, depending on circuit depth and the number of qubits.

Algorithm 2 High-level training loop for QFFN with quantum fraction α.

Require: Training data $\mathcal{D}$, quantum fraction α, model M (initialized with classical + quantum parameters), optimizer Opt

Shuffle $\mathcal{D}$ into mini-batches
for each step $= 1$ to N_{steps} **do**
  (**input**, **labels**) $\leftarrow$ BatchFrom($\mathcal{D}$)
  Opt.zeroGrad()
  logits $\leftarrow M$(**input**, α)
  loss $\leftarrow$ CrossEntropy(**logits**, **labels**)
  loss.backward() {Mixed quantum-classical gradients}
  Opt.step()
end for

B.3 Software and Hardware Environment

All experiments run on a single NVIDIA A100 40GB GPU and Intel Xeon CPUs. We use Python 3.9, PyTorch 1.13, PennyLane 0.28, and Hugging Face Transformers 4.26. Table 4 summarizes key software versions.

C Extended Experimental Results

C.1 Validation Curves Across Training Steps

Although the main text reports final perplexities, we also record validation perplexity at intermediate checkpoints (every 1k steps). QFFN stays within 0.3–0.5 perplexity points of LLaMA-7B for much of training, whereas Distilled-4B diverges more substantially after the initial warmup. This suggests that quantum sub-layers, once stabilized, do not severely impede overall convergence.

C.2 Ablation: Circuit Depth

In addition to varying the fraction of quantum layers, we experiment with circuit depth D in QFFNs and quantum attention heads. Deepening circuits from $D = 4$ to $D = 8$ typically decreases perplexity slightly (e.g., from 20.5 to 20.2), but can raise GPU memory usage and increase training time. Beyond $D = 8$, returns were minimal and training sometimes became unstable, suggesting an optimal range around 4–8 layers for our setup.

D Additional Pseudo-code and Training Details

Algorithm 2 outlines a more extensive training routine for QFFN, illustrating how quantum and classical sub-layers coexist. The parameter-shift

Hyperparameter	Value	Description
Max Sequence Length	128	Token length for both training and inference
Batch Size	8	Samples per batch (single GPU)
Learning Rate	2×10^{-5}	Initial LR for AdamW
Optimizer	AdamW	Weight decay = 0.01
Training Steps	10k	Fine-tuning steps on OpenWebText
Warmup Steps	500	LR warmup steps
Gradient Clipping	1.0	Max gradient norm
Qubit Count (n)	$\lceil \log_2(d) \rceil$	For quantum feed-forward/attention
Circuit Depth (D)	4–8	Layers of quantum gates per sub-layer
Activation Function	GELU	(Classical feed-forward only)

Table 3: Hyperparameters used for all experiments.

Component	Version/Details
Python	3.9
PyTorch	1.13
PennyLane	0.40
Transformers (HF)	4.26
CUDA	11.7
cuQuantum SDK	24.03

Table 4: Software environment details.

Model	R-1	R-2	R-L	ASL
LLaMA-7B	36.8	15.1	33.9	57.2
Distilled-4B	32.2	13.8	30.1	52.4
QFFN	35.6	14.9	33.0	56.1

Table 5: ROUGE scores on CNN/DailyMail summarization (zero-shot). ASL is the average number of tokens in generated summaries.

rule applies to quantum gates, while standard autodiff handles classical weights.

In practice, PennyLane automatically handles quantum gradients via parameter-shift, while PyTorch autograd manages classical layers. The user can toggle between `lightning.qubit` or `lightning.gpu` by specifying the device string (e.g., "lightning.gpu").

E Reproducibility Checklist

To encourage reproducibility, we release all training, nference, and evaluation scripts at https://sites.google.com/view/qnn-transformer.