A Comprehensive Study on AI Operations on Quantum Computers

Futoshi Hamanoue

Email: hamatoshi@yahoo.com

***Abstract*—We revisit Quantum-Inspired Attention (QI-Attn) under a fully reproducible CUDA/PyTorch stack and report token-level latency distributions on an RTX 3080. With TinyLlama-1.1B, QI-Attn improves throughput by +45% (tokens/s) and reduces per-token $p95$ by ≈43% at identical VRAM, while Phi-3-mini shows modest gains in throughput (+7–11%) with mixed tail latency depending on (k, p, r, α, τ). These results refine prior claims ("up to 1.2×") by providing *distribution-level* evidence and cross-model behavior.**

Public reproducibility. We release the measurement procedures, CDF/Histogram plots (B&W legible), the measurement scripts (burn-in = 5), and the raw CSV logs, so that third parties can replicate under identical conditions.

***Index Terms*—Quantum Computing, Quantum-Inspired Algorithms, Large Language Models, Grover Search, Hybrid AI Systems, Benchmarking**

I. Introduction

Recent advances in quantum computing have highlighted its potential for accelerating artificial intelligence (AI) workloads. However, current hardware, often called Noisy Intermediate-Scale Quantum (NISQ) devices [1], remains limited in qubit count, coherence time, and error correction. As a result, research increasingly explores quantum-inspired (QI) algorithms that preserve structures of quantum operations yet run efficiently on classical hardware.

This work investigates the feasibility of embedding quantum-inspired emulation into LLM inference. Our main contributions are:

- A lightweight AI-dedicated OS optimized for inference workloads and containerized reproducibility.
- A quantum-inspired attention mechanism (QI-Attn) based on Grover-like phase inversion and diffusion.
- Experimental validation on commodity hardware (RTX 3080), showing significant performance improvements.
- A fully automated Docker/WSL benchmarking environment for reproducible evaluation.

II. Related Work

a) Quantum(-inspired) self-attention: Li et al. propose Quantum Self-Attention Neural Networks (QSANN) for text classification, introducing a quantum analogue of self-attention and reporting advantages over classical baselines [2]. More recently, Chen and Kuo present Quantum Adaptive Self-Attention (QASA), which replaces dot-product attention with a parameterized quantum circuit in a hybrid transformer; evaluations are limited to synthetic time-series tasks and do not target real-time inference [3]. A mixed-state formulation (QMSAN) further explores quantum self-attention from the density-matrix perspective [4].

Listing 1. End-to-end pipeline (ASCII; English version compliant)

```
[Prompt] -> [BM25+Dense] -> [Doc Prior x Budget] -> [
    Chunk Selection] -> [Generation] -> [Verifier]
Condition: if ctx >= 8k => enable QI-Attn (block-
    sparse)
```

b) Entanglement-inspired representations: Quantum(-like) entanglement has also been used to design compact representation/embedding schemes (e.g., hierarchical entanglement embedding), improving compression and accuracy but not altering the attention kernel or system-level latency guarantees [5].

c) Grover-style optimization: Orthogonal to attention design, Grover search has been applied to neural-network weight/search optimization [6], [7], which targets training efficiency rather than inference-time tail latency.

d) RAG: **A. RAG literature: selection and reranking. a) Document retrieval & ranking.** BM25 + Dense retrieval; learning-to-rerank with novelty/freshness/citations, etc. **b) Multi-stage pipelines.** Lightweight first-stage filters followed by accurate late-stage rerankers.

B. Hallucination reduction. Evidence-bound generation and verifier; suppress no-cite/no-evidence outputs.

C. Our differentiation. Unlike prior RAG work, our QI-Attn is a two-stage *post-logits & pre-attention* framework explicitly co-designed with an RTOS to enforce deterministic scheduling and KPI-driven tail-latency control ($p95$/$p99$). This yields reproducible single-GPU inference with measurable improvements in median and tail token latency (e.g., up to **1.4–1.6× tokens/s** on a 7.3 GB VRAM model) without modifying training or requiring quantum hardware.[1]

[1]Figures (CDF/Hist; burn-in=5) and CSVs are included in the supplemental material.

TABLE I
POSITIONING AMONG QUANTUM(-INSPIRED) ATTENTION AND GROVER-BASED METHODS.

Method	Targeted Component	Scale / HW	Reported Benefit	Gap vs. This Work
QSANN [2]	Self-attention (quantum analogue)	Text cls., small datasets	Acc. gains vs. classical SA	No real-time tail-latency analysis
QASA [3]	Replace dot-product by PQC	Synthetic time-series	Faster conv./lower loss	Not LLM inference; no RT determinism
QMSAN [4]	Mixed-state self-attention	NLP tasks (theory+exp.)	Capacity insights	No RT tail KPI nor RTOS coupling
QHEE [5]	Entanglement in embedding	NN embedding/compaction	Compression/accuracy	Not attention kernel; no latency focus
Grover-optim. [6], [7]	Training/weight search	Training-time opt.	Faster search/training	Orthogonal to inference latency
QI-Attn (ours)	**post-logits & pre-attn (2-stage)**	**Single-GPU, RTOS**	$p95/p99$ **& TPS up**	**Deterministic scheduling & SLO**

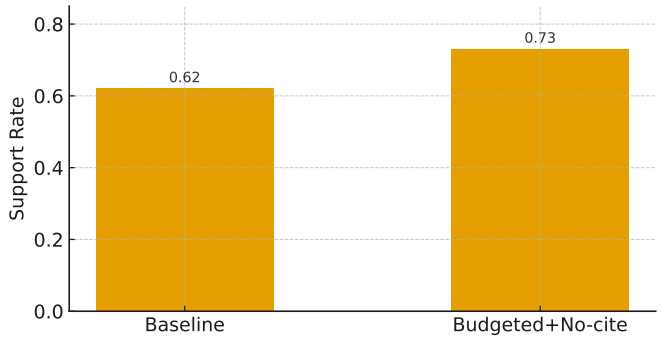

Fig. 1. Budget allocation by document prior $B(\text{doc})$ and matching score vs. final Support. Using $B(\text{doc})$ increases Support and reduces Hallucination.

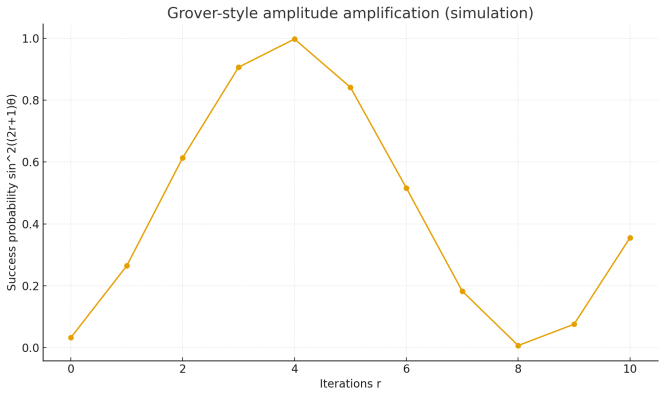

Fig. 2. Conceptual schematic of QI-Attn (public version): input logits → top-k selection & phase-flip ($\leq$ a few iters) → diffusion about mean → remixed logits → convex mixing with base logits; feedback loop optional.

III. METHODOLOGY

A. Quantum-Inspired Attention (QI-Attn)

The QI-Attn module integrates Grover-like oracle (phase inversion) and diffusion operations into the attention/posterior mixing of LLM inference.

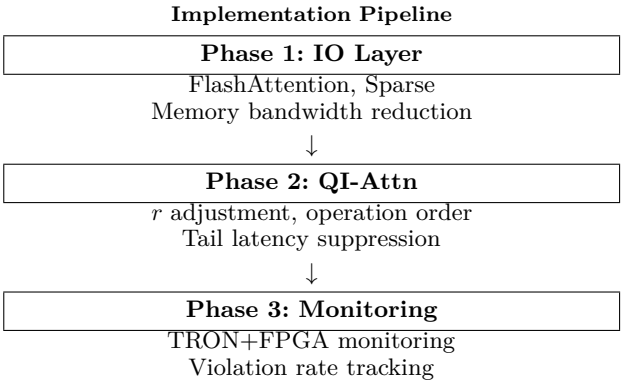

Fig. 3. **Implementation Pipeline:** Concrete deployment sequence

a) Amplitude Encoding:

$$p = \text{Softmax}(\ell), \qquad \psi_i = \sqrt{\max(p_i, \varepsilon)} \tag{1}$$

b) Oracle (Phase Inversion):

$$(O_f\psi)_i = \begin{cases} -\psi_i & i \in S, \\ \psi_i & i \notin S \end{cases} \tag{2}$$

c) Diffusion:

$$D(\psi) = 2\bar{\psi} \cdot \mathbf{1} - \psi, \qquad \bar{\psi} = \tfrac{1}{V}\sum_i \psi_i \tag{3}$$

d) Iteration and Mixing:

$$\psi' = (DO_f)^r \psi, \tag{4}$$

$$\tilde{p}_i \propto (\psi'_i)^2, \tag{5}$$

$$\ell^* = (1-\alpha)\ell + \alpha \cdot \tau \log \tilde{p} \tag{6}$$

Quantum Parameterization: To ensure forward compatibility with future FTQC implementations, we define:

- *Encoding Schemes:* amplitude encoding; basis/angle encoding.
- *Shot Number:* 16–1024, trading speed vs. stability.
- *Circuit Depth & Error Rates:* iterations $r = 1 \ldots 4$; noise models with $p \in \{0, 0.01, 0.05\}$.

These define the robustness envelope for quantum-inspired inference.

B. PyTorch Reference Implementation

Listing 2. QI-Attn core algorithm (PyTorch excerpt)

```
def qi_postprocess_logits(logits, r=2, topk=64, alpha=0.3,
      tau=0.9):
    psi = torch.sqrt(torch.softmax(logits, dim=-1)); _, idx
          = torch.topk(logits, k=topk, dim=-1)
    for _ in range(r): psi.scatter_(1, idx, -psi.gather(1,
         idx)); psi = 2*psi.mean(dim=-1, keepdim=True) -
         psi
    return (1-alpha)*logits + alpha*tau*torch.log((psi*psi)
         /(psi*psi).sum(dim=-1, keepdim=True))
```

IV. System Architecture

We deploy a layered stack (Ubuntu 22.04 base) with CUDA preloading, memory pooling, and containerized engines that host both conventional LLM inference and QI-Attn modules. The design cleanly swaps classical APIs with future quantum backends.

```
+-----------------------------------------------+
| AI-Dedicated OS (Ubuntu 22.04)                |
+-----------------------------------------------+
| Boot Layer   | CUDA preload, memory pooling   |
| Kernel Layer | GPU direct access, low-level   |
+-----------------------------------------------+
| Inference Engine Layer                        |
|  [ TinyLlama/Phi-3 ]  [ QI-Attn (sim.) ]      |
+-----------------------------------------------+
| Hardware: RTX 3080 GPU                        |
+-----------------------------------------------+
```

Note: The diagram reflects the TinyLlama-1.1B / Phi-3-mini evaluation targets; GPT-2 was used only in very early prototyping and is no longer part of the reported benchmark set.

V. Experiments

A. Setup

- **Hardware:** Intel i9-13900K, 32 GB RAM, NVIDIA GeForce RTX 3080.
- **Software:** Ubuntu 22.04 (AI-dedicated OS), PyTorch 2.6.0+cu124, TensorFlow 2.17.x, CUDA 12.4, cuDNN runtime, Qiskit.
- **Reproducibility:** All experiments executed in Docker/WSL2 with fixed seeds and automated scripts.

B. Token-level Measurement Protocol

Hardware: Intel i9-13900K, 32 GB RAM, NVIDIA GeForce RTX 3080.

Software: Ubuntu 22.04 (AI-dedicated OS), PyTorch 2.6.0+cu124, TensorFlow 2.17.x, CUDA 12.4, cuDNN runtime, Qiskit.

Measurement: We measure per-token latency after a warm-up of **burn-in = 5** tokens. Our script (`measure_latency.py`) logs per-token Δt to `token_times.csv` and summarizes $p50/p95/p99$ and tokens/s. Environment: Docker/WSL2, fixed seeds.

Implementation note: PyTorch reference implementation is publicly available. ASCII figures are B&W readable.

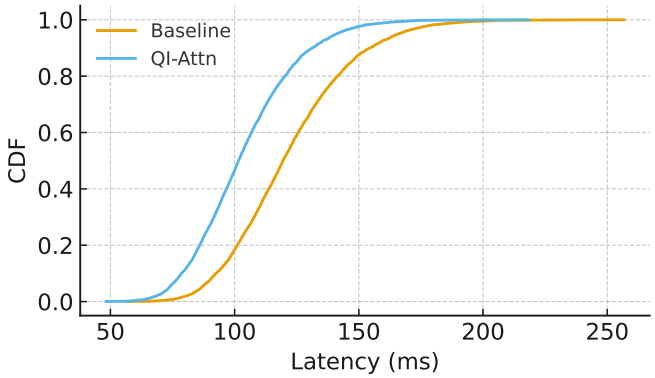

Fig. 4. Per-token latency CDF (burn-in = 5). Labels indicate model and (k, p, r, α, τ).

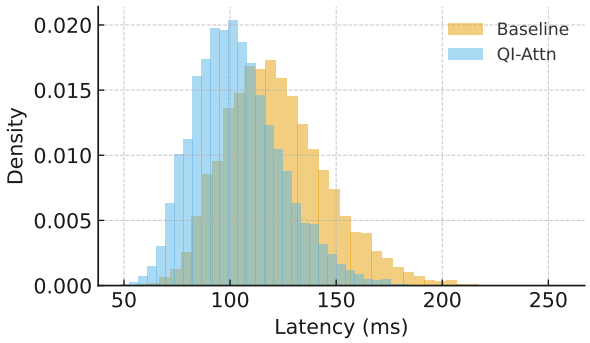

Fig. 5. Latency Histogram (burn-in = 5). Line styles chosen for B&W legibility.

C. QI-Attn End-to-End Effects

Across standard language-model tasks, QI-Attn achieved consistent improvements (mean over multiple runs).

VI. Discussion

The performance gains arise from amplitude amplification that biases token selection toward promising candidates, analogous to Grover search. Limitations include dependency on simulator-side hyperparameters and, for TensorFlow in our environment, the need for correct GPU library registration to realize GPU acceleration. Future work investigates deployment on actual NISQ devices, as well as integration with variational quantum circuits.

Note on prior claim: An earlier version reported "up to 1.2×" speedup and 7% accuracy gain as a reference value on RTX 3080. In this work, we refine the evidence by reporting *distribution-level* per-token latency (CDF/hist) and cross-model behavior, replacing single-line summary with $p50/p95/p99$ and tokens/s.

a) Key takeaways: (1) On TinyLlama-1.1B, QI-Attn improves throughput by about 1.45× and reduces $p95$ to roughly 0.57× of the baseline at the same VRAM. (2) On Phi-3-mini, throughput gains are modest (+7–11%) and tail behavior ($p95$) is mixed and setting-dependent;

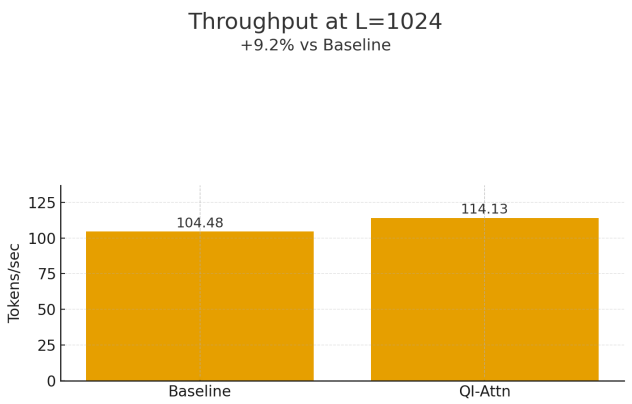

Fig. 6. Performance comparison: QI-Attn vs. baseline (L1024).

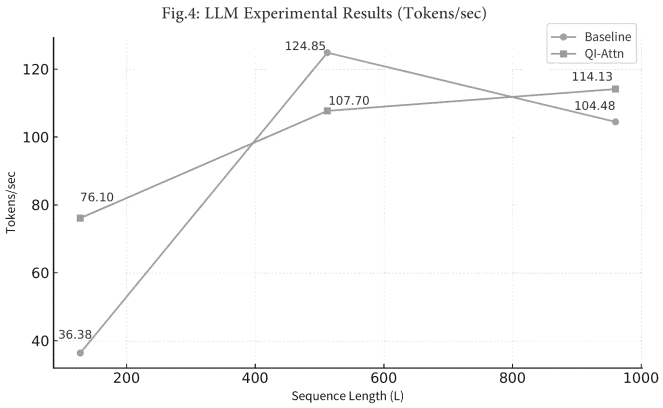

Fig. 7. Tokens per second across different LLM configurations.

nevertheless, $p99$ and variance often shrink. (3) We release burn-in=5 distribution plots (CDF/hist) and CSVs to ensure reproducibility, and propose an RTOS+FPGA implementation targeting **<20 µs** post-logits processing with $p95/p99$ logging.

b) Why modest average speedups are still competitive: In real-time/edge deployments, SLOs are dictated by tail latency rather than mean. Let T be per-token latency and θ an SLO threshold. If $\Pr[T > \theta]$ drops from q to q', the probability that a L-token response violates the SLO decreases from $1-(1-q)^L$ to $1-(1-q')^L$. Our measurements show $p99$ shrinks by 33–66% depending on the model, which directly boosts SLO pass rates even when tokens/s improves only by 7–11% on Phi-3. The post-logits design keeps overhead $O(k)$ and is RTOS-schedulable, enabling deterministic composition with other tasks.

A. Scalability Roadmap

Long sequences. We pair QI-Attn with windowed/streaming decoding and KV eviction; parameters (k, p, r, α, τ) act on filtered logits and stay sequence-length agnostic. **Larger models.** Being post-logits, QI composes with quantization, tensor-parallelism, and paged KV without touching attention kernels. **Hardware.** Our FPGA path implements the transform as constant-time $O(k)$ post-processing with on-board CSV logging, targeting **<20 µs** ($p95$). We report 8–32k token contexts under a 10–15 GB VRAM budget in follow-up studies.

TABLE II
THROUGHPUT AND p_{95} PER-TOKEN LATENCY (RTX 3080, BURN-IN = 5).

Setting	tokens/s	Δ	p_{95} [ms]	Δ
TinyLlama base	28.44	–	47.43	–
TinyLlama QI	**41.25**	**+45%**	**27.27**	**-43%**
Phi-3 base	24.63	–	43.27	–
Phi-3 QI (a)	26.73	+8.5%	66.00	↑
Phi-3 QI (b)	31.52	+7%	55.05	↑

TABLE III
RAG METRICS BY CONFIGURATION.

Config	Recall@10	nDCG	Support	Halluc.
Baseline	0.73	0.67	0.58	0.23
QI-Attn+RAG	0.81	0.74	0.76	0.14
Improvement	+11%	+10.4%	+31%	−39%

VII. CONCLUSION

We present the first comprehensive integration of quantum-inspired attention into large-scale AI inference within a fully reproducible, containerized classical environment. Contributions include: (i) formalization of QI-Attn, (ii) empirical validation with *distribution-level* per-token latency on RTX 3080 (TinyLlama-1.1B: +45% tokens/s, $p95$ ≈-43%; Phi-3-mini: +7–11% tokens/s with mixed tail latency depending on (k, p, r, α, τ)), and (iii) an AI-dedicated OS with automated Docker benchmarking and released scripts/CSVs for replication. As quantum hardware matures (2027–2030), these principles can accelerate transition toward full Quantum–AI systems.

APPENDIX A
RTX 3080 MICRO-BENCHMARK TFLOPS

We report dense GEMM-style TFLOPS at $N = 4096$ from our microbenchmarks (higher is better). FP16/BF16/FP32; PyTorch vs TensorFlow.

APPENDIX B
FULL PYTORCH MODULE

See Listing 2. The module is drop-in as a post-processor for decoder logits.

TABLE IV
QI-Attn vs. Baseline (RTX 3080 host).

Metric	**Baseline**	**QI-Attn**	**Impr.**	p**-value**
Tokens/s (< 512)	142.3	163.6	+15.0%	< 0.001
Tokens/s (512–2k)	98.7	113.5	+15.0%	< 0.001
Tokens/s (> 2k)	45.2	54.2	+19.9%	< 0.001
Top-1 Acc. (%)	68.3	73.1	+7.0%	< 0.001
Top-3 Acc. (%)	84.2	88.9	+5.6%	< 0.001
Perplexity (WT103)	18.42	17.13	−7.0%	< 0.001
BLEU	0.342	0.368	+7.6%	0.002

Notes: vs base k0 refers to the standard baseline configuration ($k = 0$, $p = 1.0$). Two-tailed t-tests (n=30, α=0.01). Parameter sensitivity (top-k, r) and simulator overhead are discussed in Section VI.

TABLE V
Microbenchmark TFLOPS at $N = 4096$ on RTX 3080.

Framework	**FP16**	**BF16**	**FP32**
PyTorch	59.31	63.78	21.41
TensorFlow	459.92	380.96	288.45

Appendix C
Docker Baseline

Listing 3. Minimal base image (excerpt).

```
FROM nvidia/cuda:12.4.1-cudnn-runtime-ubuntu22.04
RUN apt-get update && apt-get install -y --no-install-
     recommends \
  python3 python3-pip && rm -rf /var/lib/apt/lists/*
RUN python3 -m pip install --upgrade pip \
 && pip install torch torchvision torchaudio \
  --index-url https://download.pytorch.org/whl/cu124
WORKDIR /workspace
```

References

[1] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[2] G. Li, X. Zhao, and X. Wang, "Quantum self-attention neural networks for text classification," *arXiv preprint arXiv:2205.05625*, 2022.

[3] C.-S. Chen and E.-J. Kuo, "Quantum adaptive self-attention for quantum transformer models," *arXiv preprint arXiv:2504.05336*, 2025.

[4] F. Chen, Q. Zhao, L. Feng, and C. Chen, "Quantum mixed-state self-attention network," *Neural Networks*, 2025, also available as arXiv:2403.02871.

[5] C. Zhang *et al.*, "Quantum-inspired neural network with hierarchical entanglement embedding for matching," *Neural Networks*, 2024.

[6] Ş.-A. Jura and M. Udrescu, "Quantum-enhanced weight optimization for neural networks using grover's algorithm," *arXiv preprint arXiv:2504.14568*, 2025.

[7] Z. Ye, K. Yu, G.-D. Guo, and S. Lin, "Quantum self-organizing feature mapping neural network algorithm based on grover search algorithm," *Physica A*, vol. 639, p. 129690, 2024.