A Comparative Study of Classical-to-Quantum Encoding Methods Using Qrisp's QuantumFloat, Angle Encoding, and Amplitude Encoding

Abhi Muvva
Department of Physics, University of Wisconsin–Madison

This work presents a unified empirical comparison of three classical-to-quantum data encoding strategies QuantumFloat-based encoding via Qrisp, angle encoding, and amplitude encoding applied to quantum classifiers on the Iris dataset. We evaluate each method in terms of encoding time, end-to-end classification time, qubit footprint, and gate complexity. While QuantumFloat does not improve raw encoding performance and incurs significantly larger circuit sizes, its structured reversible arithmetic can provide notable speed advantages in tasks that require only a small number of quantum evaluations. When the computational workload scales and many quantum distance evaluations are needed, this benefit disappears, and the lightweight circuits of amplitude and angle encoding become more efficient. Overall, the results show that QuantumFloat is not a universal replacement for standard encodings, but a specialized mechanism that is advantageous in low-evaluation, arithmetic-heavy settings and inefficient in broader, high-repetition quantum ML pipelines. This study offers a unified framework for analysing such trade-offs and clarifies when high-level quantum arithmetic abstractions are practically useful.

I. INTRODUCTION

High-level quantum programming frameworks aim to reduce the burden of writing complex quantum algorithms. Qrisp, in particular, introduces abstractions such as `QuantumVariable`, `QuantumArray`, and `QuantumFloat` that resemble classical programming constructs while maintaining full circuit compilability. These abstractions raise an important question: *Can high-level representations of numerical data also serve as effective classical-to-quantum data encodings for machine learning workloads?*

Traditional encoding methods such as amplitude or angle encoding require careful normalization, feature rescaling, and hand-built circuits. While powerful, these techniques can complicate the integration of classical data into quantum pipelines. In contrast, Qrisp's `QuantumFloat` promises a more direct mechanism for representing classical numbers with fixed-point precision. Motivated by this possibility, this project explores whether QuantumFloat encoding provides practical advantages; particularly in ML workloads.

We conduct a detailed study on the Iris dataset. The central objective is to evaluate simulation time, resource consumption, and classification accuracy, and to determine whether QuantumFloat-based workflows meaningfully reduce overhead in gate construction, arithmetic, and workflow engineering.

II. BACKGROUND

A. Angle Encoding

Angle encoding is a classical-to-quantum data mapping strategy in which real-valued classical features are embedded into quantum states through rotation gates. Each classical feature value x_i is first mapped to a rotation parameter $\theta_i = f(x_i)$, and then applied to a qubit using single-qubit rotations such as $R_x(\theta)$, $R_y(\theta)$, or $R_z(\theta)$. As described in [1], the probability amplitudes of the resulting quantum state depend directly on these rotation angles, allowing classical information to be encoded naturally into the geometry of the Bloch sphere.

For a single qubit initialized in $|0\rangle$, applying a rotation $R_y(\theta)$ yields the encoded quantum state

$$R_y(\theta)|0\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + \sin\left(\frac{\theta}{2}\right)|1\rangle,$$

with analogous expressions for $R_x(\theta)$ and $R_z(\theta)$. Thus, each classical feature determines the relative weighting of the computational basis states. For an n-dimensional feature vector $x = (x_1, \ldots, x_n)$, the full angle-encoded quantum state is constructed by applying a local rotation to each qubit:

$$|\psi(x)\rangle = \bigotimes_{i=1}^{n} R_y(\theta_i)|0\rangle.$$

Since each qubit stores exactly one classical feature, angle encoding scales linearly with the dimension of the classical input and is notably hardware-efficient for near-term quantum devices. This simplicity makes angle encoding widely used in parameterized quantum circuits and hybrid quantum–classical machine learning models, where low circuit depth and robustness to noise are essential [1].

B. Amplitude Encoding

Amplitude encoding embeds classical data directly into the probability amplitudes of a quantum state. For a classical vector $x = (x_1, \ldots, x_N)$, the amplitude-encoded quantum state is defined as

$$|x\rangle = \sum_{i=1}^{N} \frac{x_i}{\|x\|}|i\rangle, \qquad \|x\|^2 = \sum_{i=1}^{N} |x_i|^2.$$

This construction requires $\log_2(N)$ qubits, enabling an exponentially compact representation: N classical components may be stored using only $\log_2(N)$ quantum degrees of freedom. This normalization requirement ensures that the resulting quantum state is physically valid, while the mapping allows quantum algorithms to exploit interference and superposition over all components simultaneously [1].

To illustrate, consider a four-dimensional vector $x = (x_1, x_2, x_3, x_4)$. Two qubits suffice to represent the amplitude-encoded state:

$$|x\rangle = \frac{x_1}{\|x\|}|00\rangle + \frac{x_2}{\|x\|}|01\rangle + \frac{x_3}{\|x\|}|10\rangle + \frac{x_4}{\|x\|}|11\rangle.$$

This encoding is particularly expressive because it embeds the entire classical vector into the global structure of the quantum state [1]. However, preparing such amplitude encoded states generally requires deeper and more complex circuits compared to angle encoding, making implementation challenging on present-day noisy intermediate-scale quantum (NISQ) hardware. Despite this, amplitude encoding remains a theoretically powerful representation for quantum machine learning, especially for algorithms exploiting high-dimensional vector operations such as inner products and kernel evaluations.

C. Qrisp's QuantumFloat

The `QuantumFloat` type [2] is Qrisp's fixed-point numerical representation, enabling real-valued computation on quantum hardware while remaining fully reversible. A `QuantumFloat` with mantissa size m, exponent e, and an optional sign bit represents numbers of the form

$$x = s \cdot \left(\sum_{k=0}^{m-1} b_k 2^{-(k+1)} \right) \cdot 2^e,$$

where $b_k \in \{0, 1\}$ denote the mantissa qubits and $s \in \{\pm 1\}$ is present only for signed floats. Thus the representable values form a uniformly spaced grid with resolution 2^{e-m}. Encoding a classical value x into a `QuantumFloat` corresponds to preparing its binary fixed-point expansion across the mantissa and sign qubits. More generally, supplying a dictionary of amplitudes to the slice operator (e.g., `qf[:] = {x_1:\alpha_1, x_2:\alpha_2}`) prepares the superposition

$$|\psi\rangle = \sum_i \alpha_i \, |x_i\rangle,$$

where each $|x_i\rangle$ is the binary-encoded fixed-point representation of x_i. This makes `QuantumFloat` a genuine quantum data type capable of storing and manipulating coherent numerical superpositions.

Arithmetic operations such as addition, subtraction, multiplication, and comparisons are implemented by operator overloading, each compiling to a reversible circuit composed of controlled additions, ripple-carry or carry-lookahead adders, reversible multipliers, and comparators [2]. For instance, multiplying two floats a and b yields a new register c representing

$$|a\rangle|b\rangle|0\rangle \xrightarrow{U_{\text{mult}}} |a\rangle|b\rangle|ab\rangle,$$

where U_{mult} is a reversible multiplication circuit that preserves coherence over all superposed inputs. Since these operations generate temporary work registers, Qrisp integrates an automated uncomputation mechanism based on the Unqomp algorithm [3, 4], ensuring that ancilla qubits are disentangled and returned to $|0\rangle$ after use. This maintains reversibility and prevents entanglement leakage that would otherwise accumulate during chained arithmetic.

Although not a traditional feature encoding scheme like angle or amplitude encoding, `QuantumFloat` implicitly encodes classical numerical data in binary fixed-point amplitudes and permits coherent arithmetic on them. This makes it uniquely suited for quantum algorithms that require explicit numerical computation; such as distance evaluation, modular arithmetic, and Grover oracles; while preserving mathematical correctness, reversibility, and compatibility with high-level algorithmic structure.

III. METHODS

This project began with the goal of comparing quantum data–encoding strategies for classification, beginning with variational quantum circuits (VQCs) and later transitioning to KNN-based algorithms. The workflow evolved through multiple algorithmic choices, each influenced by the practical limitations observed during implementation. Below, we outline this progression in detail.

A. VQC with Angle Encoding for the Iris Dataset

Data Encoding

The Iris dataset was reduced to two features (petal length and petal width) and normalized to the interval $[0, 1]$ using `MinMaxScaler`. For angle encoding, each feature value x was shifted to $x - 0.5$, mapping the domain to $[-0.5, 0.5]$, which was then linearly scaled to rotation angles in $[-\pi, \pi]$ for use in parametrized gates.

Variational Circuit Structure

The VQC model followed the general structure of the Pennylane variational-classifier demo, but was significantly extended to support multiclass classification. Each layer of VQC consisted of:

- a set of `ROT` gates, each parameterized by three trainable angles;
- entangling `CNOT` gates to generate correlations between the two encoded features;
- a trainable weight vector whose dimensionality grew as $3 \times n_{\text{layers}}$.

The forward pass evaluated the circuit for each data point, and model parameters were optimized with gradient-based updates analogous to classical neural-network backpropagation. Classification was performed using log-odds interpolation to map measurement outcomes to class probabilities.

B. Precision Constraints and QuantumFloat Encoding in Qrisp

A shift to Qrisp introduces a new constraint: fixed-point encoding inherently rounds real values to the closest representable point. Thus, classification performance depends heavily on the chosen bit-precision. QuantumFloats of width $k \in \{5, 6, 7, 8\}$ were tested by analyzing distortion through correlation matrices and covariance-difference heatmaps. Empirically, $k = 5$–6 bits provided the best balance between precision and qubit cost.

However, the more restrictive limitation arises from Qrisp's compilation model. Variational quantum classifiers rely almost entirely on parametrized rotation gates such as $R_X(\theta)$, $R_Y(\theta)$, $R_Z(\theta)$, which are cheap, native primitives in gate-level frameworks. Their performance comes from manipulating these rotations directly.

Qrisp, however, provides a very different advantage: high-level reversible arithmetic through QuantumFloat registers. Additions, multiplications, and comparisons are compiled automatically into reversible circuits, allowing fixed-point algorithms to be expressed concisely and efficiently. A VQC does not meaningfully use this functionality, since its logic is rotation-based, not arithmetic-based.

Implementing a VQC in Qrisp therefore discards the main benefit of the framework. Instead of leveraging QuantumFloat arithmetic, it forces Qrisp to emulate gate-level variational layers, offering no structural or resource advantage. Qrisp is thus better applied to classifiers whose core computation *is* arithmetic where its fixed-point abstractions naturally map to efficient reversible circuits.

C. Nearest-Centroid and KNN Classification Using Qrisp

Given these architectural constraints, the natural direction was to adopt classification models whose core computations *are* arithmetic. This led to nearest-centroid and k-NN methods, where QuantumFloat operations directly express the required distance evaluations and comparisons.

1. Initial Centroid-Based Design

The first Qrisp prototype used a nearest-centroid classifier:

1. Compute classical centroids for the three Iris classes.
2. Encode each centroid as a `QuantumFloat` with precision $k = 5$.
3. For each test sample, encode its two features and compute Euclidean distances to all three centroids.
4. Select the class whose centroid produced the minimum distance.

Before settling on the final design, a toy Euclidean-distance prototype was implemented to inspect resource growth under Qrisp's arithmetic model. As expected, evaluating Euclidean distance required multiple QuantumFloat multiplications, and the associated reversible logic rapidly inflated the qubit count. The compiled circuits for $k = 5$–8 bits already showed a steep, linear increase in total qubits per distance call, as illustrated in Fig. 1.

This made one point immediately clear: although Qrisp provides powerful arithmetic abstractions, exploiting them fully comes at a substantial resource cost. Using Euclidean distance would force the classifier into a regime where arithmetic depth and ancilla demand dominate, offering no practical scalability. To avoid committing to a purely arithmetic approach and to retain resource efficiency, the distance metric itself had to be redesigned.

2. Optimized Hamming-Distance Version

To address the Euclidean-distance bottleneck, the distance computation was redesigned to avoid QuantumFloat multiplications entirely:

1. After normalizing each feature to $[0, 1]$, multiply by 2^k before encoding; this preserves precision without requiring fractional arithmetic.
2. Replace Euclidean distance with Hamming distance between the fixed-point bitstrings, eliminating multiplications entirely.
3. Perform each distance computation inside an isolated Qrisp session, which scopes all ancilla registers locally so they are discarded after compilation.

A more detailed explanation of this transformation is given below, showing how integer scaling allows Hamming distance to approximate Euclidean distance while substantially reducing reversible-circuit resources.

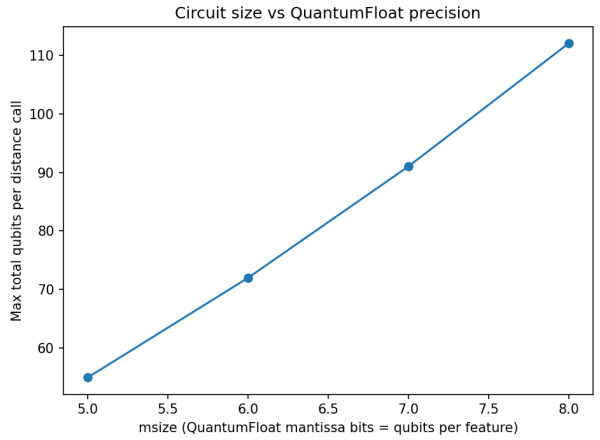

FIG. 1: Growth of total qubits required for a single Euclidean-distance computation as a function of QuantumFloat precision k. The rapid increase reflects the ancilla and reversible-arithmetic overhead introduced by multiplications in the Euclidean metric.

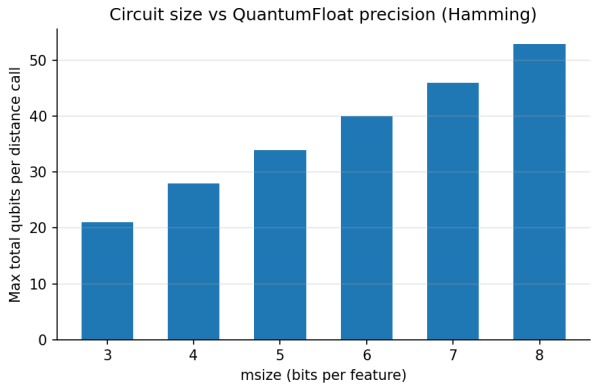

FIG. 2: Total qubits required for computing Hamming distance as a function of QuantumFloat precision k. Compared to the Euclidean-distance version (Fig. 1), the qubit count is reduced by roughly one-half, reflecting the elimination of multiplication-heavy reversible arithmetic.

a. Fixed-point scaling and bitstring encoding. Each normalized feature $x \in [0, 1]$ is mapped to an integer

$$x_{\text{int}} = \lfloor x \cdot 2^k \rceil ,$$

so that it can be stored exactly in a k-bit QuantumFloat register. This avoids fractional arithmetic entirely and preserves relative spacing, since

$$|x - y| \approx 2^{-k} \, |x_{\text{int}} - y_{\text{int}}|.$$

b. Replacing Euclidean distance with Hamming distance. The Euclidean metric on two-dimensional features,

$$d_{\text{Euc}}(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2},$$

requires QuantumFloat multiplications and squaring operations, each of which introduces many ancillas and deep reversible subcircuits. After integer scaling, we instead compare the binary expansions:

$$d_{\text{Ham}}(x_{\text{int}}, y_{\text{int}}) = \sum_{j=0}^{k-1} \mathbf{1}\left[x_{\text{int}}^{(j)} \neq y_{\text{int}}^{(j)} \right].$$

c. Resource advantage. Because Hamming distance uses only bitwise comparisons, it compiles to significantly shallower reversible circuits with far fewer ancillas than Euclidean distance. This explains the near factor-of-two reduction in qubits observed in Fig. 2.

Now the next step is to extend the algorithm from nearest-centroid ($k = 1$) to full k-nearest-neighbour voting. This design trades runtime for accuracy.

D. Quantum KNN in Qiskit

Qiskit qKNN Pipeline Overview

The Qiskit implementation mirrors the structure of the Qrisp classifier but replaces QuantumFloat arithmetic with a fidelity-based quantum kernel. The pipeline is:

1. Encode each sample into a two-qubit quantum state using one of two schemes: angle encoding or per-feature amplitude encoding.

2. For each test sample, compute similarities to all N_{train} training samples using the SWAP test.

3. Select the top-k highest similarities and assign the label by majority vote.

The SWAP test provides a natural distance-like quantity. Given two quantum states $|\psi\rangle$ and $|\phi\rangle$, the ancilla measurement yields

$$P(1) = \frac{1 - |\langle\psi|\phi\rangle|^2}{2}, \tag{1}$$

so the inner-product similarity is

$$S(\psi, \phi) = |\langle\psi|\phi\rangle|^2 = 1 - 2P(1), \tag{2}$$

and the qKNN kernel is defined directly by this fidelity.

Angle Encoding

In the angle-encoded version, each normalized feature $x_j \in [0, 1]$ is mapped to a single-qubit rotation:

$$\theta_j = \pi x_j, \qquad |q_j\rangle = R_y(\theta_j)\,|0\rangle = \cos\left(\frac{\theta_j}{2}\right)|0\rangle + \sin\left(\frac{\theta_j}{2}\right)|1\rangle. \tag{3}$$

For two features, the resulting representation is the product state

$$|x\rangle = |q_1\rangle \otimes |q_2\rangle. \tag{4}$$

These states are prepared directly by applying two R_y rotations to an initial $|00\rangle$. Training samples are encoded once into angle vectors; test samples are encoded on demand inside each SWAP-test circuit.

Per-Feature Amplitude Encoding

In the amplitude version, each feature is embedded directly into the computational-basis amplitudes:

$$|q_j\rangle = \sqrt{x_j}\,|0\rangle + \sqrt{1-x_j}\,|1\rangle, \qquad x_j \in [0,1]. \tag{5}$$

The full feature state is again a separable two-qubit product:

$$|x\rangle = |q_1\rangle \otimes |q_2\rangle, \tag{6}$$

constructed explicitly as a statevector and loaded using Qiskit's `initialize` instruction. Although the preparation circuit differs from the angle-encoding case, the final representation is structurally identical: one qubit per feature, encoded as a local, normalized two-level state.

Unified SWAP-Test Classification

Because both encoding schemes produce two-qubit product states $|x\rangle = |q_1\rangle \otimes |q_2\rangle$, the downstream classification pipeline is unchanged:

1. Prepare $|\psi_{\text{test}}\rangle$ and $|\phi_i^{(\text{train})}\rangle$ on separate registers.
2. Apply the SWAP test to estimate the fidelity $S_i = |\langle\psi|\phi_i\rangle|^2$.
3. Select the k largest similarities and vote.

Thus, angle encoding and amplitude encoding differ only in the *state-preparation layer*; the quantum kernel, similarity computation, and kNN structure remain identical.

IV. RESULTS

A. Encoding Time

Encoding times across all methods are summarized directly in Table I. QuantumFloat does not outperform amplitude encoding: for every tested precision, the QuantumFloat encoding time lies between 2.0×10^{-4} s and 3.9×10^{-4} s, whereas amplitude encoding completes in 3.65×10^{-4} s. Angle encoding remains the fastest overall, with an encoding time of 6.53×10^{-5} s.

Thus, QuantumFloat consistently sits between amplitude and angle encoding: slower than amplitude, faster than angle. Importantly, its encoding cost grows only mildly with m, indicating that register width, rather than arithmetic depth, dominates Qrisp's compilation overhead in this step.

B. Classification Time

Classification-time behavior diverges sharply across tasks, as shown in Table I.

Centroid-based CNN–QF. QuantumFloat is significantly faster than both amplitude and angle encodings. The CNN–QF classification time ranges from 0.108 s (for $m = 3$) to 0.587 s (for $m = 8$), whereas amplitude and angle methods require 3.15 s and 1.98 s, respectively. This speedup arises from a fundamental asymmetry: the QuantumFloat centroid classifier evaluates only *three* distances per test sample (one per class), while the amplitude- and angle-based pipelines must perform on the order of ~ 100 SWAP-test or overlap-based evaluations. Thus, the arithmetic cost of QF is amortized over far fewer calls, making it substantially faster in this setting.

kNN–QF. For kNN, the trend reverses: QuantumFloat becomes progressively slower. Classification time increases from 4.45 s at $m = 3$ to 23.8 s at $m = 8$, far exceeding amplitude and angle methods. Here, every method must compute a distance to all 112 training samples, eliminating the CNN asymmetry. Under this uniform workload, the heavier reversible-arithmetic circuits of QuantumFloat dominate the runtime, whereas amplitude and angle encodings rely on lighter SWAP-test-style primitives.

Thus, QuantumFloat excels in centroid-style classification, where only a few distances are computed, but is not competitive in full kNN pipelines where distance computations scale with dataset size.

C. Accuracy

Beyond runtime, Table I also highlights a clear accuracy contrast between the two QF pipelines. The CNN–QF classifier exhibits noticeably lower accuracy (0.60–0.72 across m), reflecting the coarse decision bound-

TABLE I: Unified comparison of amplitude, angle, and QuantumFloat-based CNN/kNN classifiers across accuracy, encoding time, classification time, and resource usage.

Method	m	Accuracy	Enc. Time (s)	Class. Time (s)	Qubits	Depth	1q Gates	2q Gates	$>$ 2q Gates
Amplitude	–	0.94	3.65×10^{-4}	3.15	5	5	2	2	2
Angle	–	0.92	6.53×10^{-5}	1.98	5	5	6	0	2
CNN–QF	3	0.72	2.00×10^{-4}	0.108704	21	38	51	49	0
	4	0.6267	2.53×10^{-4}	0.196914	28	53	86	81	0
	5	0.6333	2.55×10^{-4}	0.249769	34	60	107	101	0
	6	0.68	2.72×10^{-4}	0.314680	40	69	129	121	0
	7	0.60	3.21×10^{-4}	0.402280	46	76	150	141	0
	8	0.6067	2.91×10^{-4}	0.586922	53	91	205	193	0
kNN–QF	3	0.9211	2.42×10^{-4}	4.449878	21	38	51	49	0
	4	0.9211	2.85×10^{-4}	8.154689	28	53	86	81	0
	5	0.9211	2.59×10^{-4}	10.383830	34	60	107	101	0
	6	0.8947	3.28×10^{-4}	12.591710	40	69	129	121	0
	7	0.9211	3.06×10^{-4}	16.488780	46	76	150	141	0
	8	0.8684	3.87×10^{-4}	23.840250	53	91	205	193	0

ary imposed by centroid-based classification. In contrast, kNN–QF maintains high accuracy (typically 0.89–0.92), matching the amplitude and angle baselines. Thus, while CNN–QF benefits from extremely low classification times, its predictive performance is limited, whereas kNN–QF delivers strong accuracy at the cost of substantially higher computational overhead.

D. Resource Estimates

Raw circuit resources for all methods are listed in Table I, but these values represent the cost of a *single* circuit execution. The effective resource footprint per classification therefore depends critically on how many times each method must call its distance-evaluation subroutine.

Amplitude and angle encodings use extremely small circuits (5 qubits, depth 5, with only a handful of gates), but these subroutines must be executed ~ 100 times.

In contrast, QuantumFloat circuits are far wider and deeper: 21–53 qubits, depths of 38–91, and orders-of-magnitude more gates. But CNN–QF requires only *three* evaluations per test sample, while kNN–QF requires 112.

Thus, the practical resource trade-off is not determined by circuit size alone but by the product of circuit size and invocation count. QuantumFloat trades hardware efficiency for arithmetic expressiveness: each circuit is expensive, but the CNN variant compensates by requiring very few distance calls. kNN–QF lacks this advantage, and its large circuits must be repeated 112 times, yielding the highest overall overhead among all methods.

In short, QuantumFloat incurs steep per-circuit qubit and gate costs, but the centroid-based model amortizes this cost over very few calls, enabling favorable performance despite heavy individual circuits. Amplitude and angle encodings enjoy minimal circuit footprints but accumulate substantial total resource usage due to repeated SWAP-test evaluations.

E. Fidelity–Space–Time Comparison

To compare methods on equal footing, we adopt an FST-style efficiency metric inspired by the resource trade-off framework of [5]. For each method i, we compute:

- Classification fidelity F_i from the confusion matrix,
- Circuit space $S_i = Q_i \times G_i$, where Q_i is the qubit count and G_i is the total gate count per circuit,
- Total classification time T_i, which *already* includes the multiplicative cost of repeatedly invoking the distance circuit (3 calls for CNN–QF, ~ 100 for amplitude/angle centroid classifiers, and 112 for all kNN variants).

We then form the composite efficiency score

$$\text{Score}_i = \frac{F_i}{S_i\, T_i},$$

which rewards methods that maintain high accuracy while using fewer circuit resources and completing classification rapidly.

This combined perspective reveals a structured trade-off. Low-precision CNN–QF models achieve the highest FST efficiency: although their circuits are large, they are executed only three times and produce competitive fidelity at extremely low classification times. Amplitude and angle encodings incur minimal circuit cost per invocation but suffer from the large number of repeated overlap-based evaluations, reducing their overall score. kNN–QF achieves strong fidelity but is penalized by its heavy circuit footprint and the need to run it 112 times per test sample.

Overall, these results indicate that QuantumFloat should not be viewed as a competitive quantum data en-

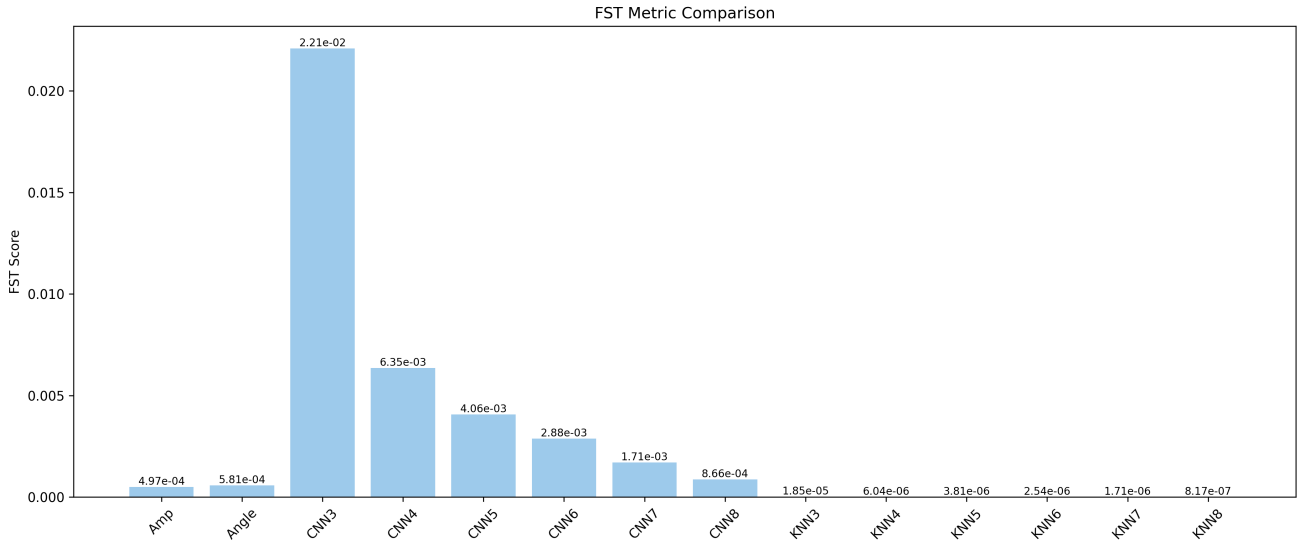

FIG. 3: FST score across all encoding and model variants. Higher scores indicate more efficient classifiers, combining accuracy, circuit size, and runtime. Low-precision QuantumFloat CNN circuits achieve the best overall efficiency.

coding scheme: although it achieves strong efficiency in the special case of three-way centroid classification, its performance in general settings is comparable to or worse than amplitude and angle encodings while requiring substantially larger qubit counts, deeper circuits, and higher gate overhead.

V. DISCUSSION

Q1. Which kinds of algorithms meaningfully benefit from using `QuantumFloat`-based encoding in Qrisp?

Only algorithms whose computation is fundamentally fixed-point arithmetic gain an advantage from `QuantumFloat`. Operations such as distance evaluations, comparisons, accumulations, and simple numeric transforms align with Qrisp's reversible–arithmetic compilation and therefore produce structured, efficient circuits.

Algorithms whose power comes from bespoke gate layouts or variational unitaries do not benefit in the same way. They can still be implemented in Qrisp, but `QuantumFloat` offers no structural or resource advantage; Qrisp behaves essentially as a standard gate-construction interface.

Q2. Why was Euclidean distance abandoned in favor of a discrete metric?

As discussed earlier in III C 1, a small Euclidean-distance prototype was implemented to assess how Qrisp's reversible-arithmetic model scales with precision. Even for the 2D Iris features, evaluating

$$\|x - \mu_k\|_2^2 = (x_1 - \mu_{k,1})^2 + (x_2 - \mu_{k,2})^2 \tag{7}$$

requires two fixed-point subtractions, two multiplications, and a subsequent accumulation. In Qrisp, each of these operations expands into a structured reversible circuit incorporating carry-propagation logic and the corresponding ladder of ancilla qubits. As a consequence, the total qubit count per distance call grew steeply with bit-precision: for $k = 5$–8 bits, the compiled circuits already exhibited a nearly linear increase in allocated qubits and overall gate count, as illustrated in Fig. 1.

This experiment revealed a clear limitation. Although Qrisp's arithmetic abstractions are expressive, relying fully on reversible multipliers and adders pushes the classifier into a regime where ancilla demand and arithmetic depth dominate the resource profile. In this setting, Euclidean distance is not merely inefficient, it fundamentally prevents the method from scaling to higher precision without overwhelming the qubit budget. To preserve practicality and avoid committing to a heavy arithmetic pipeline, the distance metric itself had to be redesigned, prompting the shift to a discrete Hamming-style comparison on fixed-point bitstrings.

VI. CONCLUSION

This work compared amplitude, angle, and QuantumFloat-based encodings across accuracy, runtime, and circuit resources. The findings show that `QuantumFloat` is not a generally superior encoding method; its advantages appear only when the number of distance evaluations is intrinsically small.

QuantumFloat circuits are far larger than those of amplitude or angle encoding, but their reversible-arithmetic structure enables very fast execution when invoked only a few times. Thus, the CNN–QF pipeline achieves excellent efficiency: three distance calls are sufficient, yielding low classification times and the highest overall FST performance.

In contrast, kNN–QF exposes the core limitation. When 112 distance evaluations are required, the heavy QuantumFloat circuits dominate runtime, making amplitude and angle encodings decisively more practical due to their compact SWAP-test constructions.

Accuracy follows this trend: kNN–QF matches the baselines, while CNN–QF is limited by centroid geometry rather than quantum encoding.

Overall, QuantumFloat should be treated as a targeted tool: valuable for low-call, arithmetic-heavy subroutines, but inefficient when distance computations scale with dataset size.

CODE AVAILABILITY

All implementations used in this work—including the amplitude-encoding, angle-encoding, centroid-based QuantumFloat (CNN–QF), and kNN–QF classifiers, along with the resource–estimation scripts—are available at `https://github.com/Abhi-Muvva/qrisp-quantumfloat-evaluation`.

[1] M. Rath and H. Date, arXiv preprint (2023), arXiv:2311.10375 [quant-ph].

[2] R. Seidel, N. Tcholtchev, S. Bock, and M. Hauswirth, arXiv preprint (2022), arXiv:2406.14792 [quant-ph].

[3] T. Paradis, J. Bausch, M. Schüler, and B. Douglas, arXiv preprint (2019), arXiv:1904.00633.

[4] R. Seidel, N. Tcholtchev, S. Bock, and M. Hauswirth, Lecture Notes in Computer Science 10.1007/978-3-031-38100-$3_1$1 (2023), arXiv:2307.11417 [quant-ph].

[5] D. Litinski, Quantum **3**, 128 (2019), arXiv:1808.02892 [quant-ph].