Verlässliche und Skalierbare Softwaresysteme
Dependable and Scalable Software Systems

Leibniz Universität Hannover

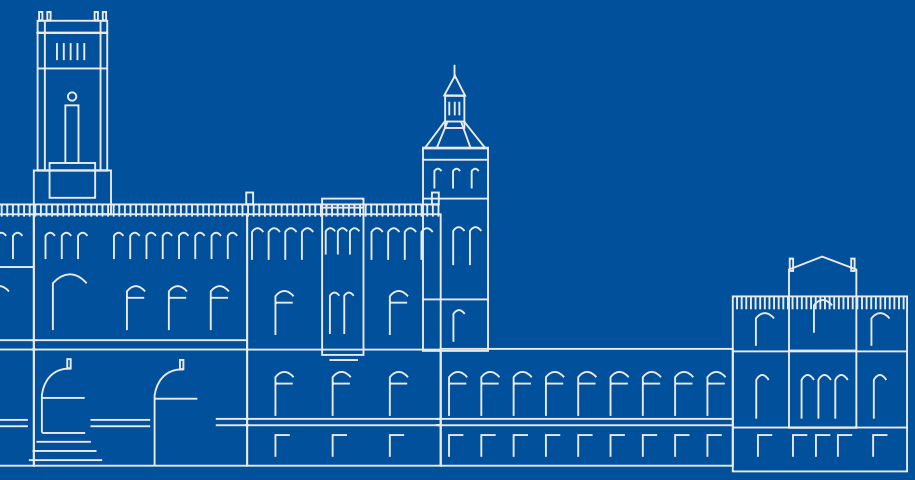
Christian Jesinghaus

Product Quantization Techniques for Accelerating Vector Database Operations on a Hybrid Quantum Computing Machine

Bachelor's Thesis in Computer Science

4. September 2025

Please cite as:
Christian Jesinghaus, "Product Quantization Techniques for Accelerating Vector Database Operations on a Hybrid Quantum Computing Machine" Bachelor's Thesis, Leibniz University Hannover, Institute for Systems Engineering, September 2025.



Leibniz University Hannover
Institute for Systems Engineering
Dependable and Scalable Software Systems
Appelstr. 4 · 30167 Hannover · Germany

Product Quantization Techniques for Accelerating Vector Database Operations on a Hybrid Quantum Computing Machine

Bachelor's Thesis in Computer Science

vorgelegt von

Christian Jesinghaus

geb. am 2. March 1999
in Hannover

angefertigt am

Institut für Systems Engineering
Fachgebiet Verlässliche und Skalierbare Softwaresysteme

Fakultät für Elektrotechnik und Informatik
Leibniz Universität Hannover

Erstprüfer:	**Prof. Dr. Jan S. Rellermeyer**
Zweitprüfer:	**Prof. Dr.-Ing. habil. Daniel Lohmann**
Betreuer:	**Prof. Dr. Jan S. Rellermeyer**

Beginn der Arbeit:	**May 8th 2025**
Abgabe der Arbeit:	**September 8th, 2025**

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Zusatz: Hinweis zum Einsatz von KI-Assistenzwerkzeugen

Hiermit erkläre ich, dass ich bei der Anfertigung der vorliegenden Arbeit KI-gestützte Assistenzwerkzeuge verwendet habe, insbesondere GitHub Copilot (in Visual Studio Code) und ChatGPT (OpenAI). Die Werkzeuge dienten mir als Inspirationsquelle sowie als peer-ähnliche, interaktive Unterstützung für die Ideenfindung, die Erörterung von Lösungsansätzen und Rückmeldungen zu Code und Text. Darüber hinaus nutzte ich sie zur Unterstützung bei Formatierung, Formulierungsvarianten und Korrekturlesen. Eine autonome Erstellung und unveränderte Übernahme von Textpassagen, Code, Analysen oder Struktur war nicht Ziel der Nutzung. Die übernommenen Inhalte wurden von mir geprüft, angepasst und in den Gesamtkontext der Arbeit integriert.

Declaration

I declare that the work is entirely my own and was produced with no assistance from third parties. I certify that the work has not been submitted in the same or any similar form for assessment to any other examining body and all references, direct and indirect, are indicated as such and have been cited accordingly.

Addendum: Note on the Use of AI Assistance

I hereby declare that, in preparing this thesis, I used AI-assisted tools, in particular GitHub Copilot (in Visual Studio Code) and OpenAI's ChatGPT. These tools served as sources of inspiration and provided peer-like, interactive support for ideation, discussing solution approaches, and feedback on code and text. I also used them for support with formatting, phrasing options, and proofreading. Autonomous generation or verbatim inclusion of text passages, code, analyses, or structure was not the aim of their use. Any incorporated material was reviewed, adapted, and integrated by me into the overall context of the thesis.

(Christian Jesinghaus)
Hannover, 4. September 2025

ABSTRACT

Modern vector search systems often use approximate nearest neighbor (ANN) techniques such as Product Quantization (PQ) to meet latency and memory constraints at scale. While PQ preserves neighborhood structure with compact codes, two bottlenecks remain: evaluating many distances during training and querying, and training subspace codebooks via k-means. This thesis investigates whether PQ can be reformulated so that core steps benefit from quantum primitives and, if so, whether this yields theoretical speedups without breaking PQ's lookup-and-accumulate structure. For that the thesis introduces: (i) an *additive* log-fidelity objective, $d_{\ln}(\psi, \phi) = -\ln\big(|\langle\psi|\phi\rangle|^2 + \varepsilon\big)$, which aligns with PQ's per-partition decomposition, and (ii) a *safeguarded* centroid update: a majorize–minimize surrogate yielding a Rayleigh–Ritz candidate (principal eigenvector of a weighted covariance), with a projected Riemannian gradient fallback on the unit sphere that ensures monotone descent under mild assumptions. Distances are estimated via amplitude encoding and SWAP tests. Grover/Dürr–Høyer minimum finding and QRAM are treated as theoretical upgrades.

On the 64-D *Digits* dataset (fixed hyperparameters), the prototype achieves accuracy comparable to a classical PQ baseline. With 300 training samples both pipelines reach 95% accuracy, and per-sample clustering losses remain stable. Limitations are dominated by runtime and the absence of real quantum memory/hardware and the centroid update which is costlier than a Euclidean mean, but necessary under these circumstances. To the best of the authors knowledge, this is the first formulation of PQ within a quantum-information setting. The results suggest viability and outline a pathway to practical gains once QRAM, quantum minimum finding, and quantum spectral routines become available. Short-term improvements could include variance reduction, parallelization across partitions, and hyperparameter tuning.

The source code concerning the implementations can be accessed at https://github.com/ChristianJesinghaus/Hybrid-Quantum-Classical-Product-Quantization-for-Vector-Databases.

KURZFASSUNG

Moderne Vektor-Suchsysteme verwenden häufig Approximate Nearest Neighbor (ANN)-Techniken wie Product Quantization (PQ), um Latenz- und Speicherbeschränkungen im großen Maßstab einzuhalten. Während PQ die Nachbarschaftsstruktur mit kompakten Codes bewahrt, bleiben zwei Engpässe bestehen: die Auswertung vieler Distanzen während des Trainings und der Abfrage sowie das Trainieren von Subraum-Codebüchern mittels k-means. Diese Arbeit untersucht, ob PQ so umformuliert werden kann, dass zentrale Schritte von Quanten-Primitiven profitieren und ob dies zu theoretischen Beschleunigungen führt, ohne PQs Lookup-und-Akkumulations-Struktur zu zerstören. Dafür führt die Arbeit ein: (i) ein *additives* Log-Fidelity-Objektiv, $d_{\ln}(\psi,\phi) = -\ln\big(|\langle\psi|\phi\rangle|^2 + \varepsilon\big)$, das mit PQs per-partition-Zerlegung übereinstimmt, und (ii) ein *abgesichertes* Zentroid-Update: ein Majorize–Minimize-Surrogat, das einen Rayleigh–Ritz-Kandidaten (Haupt-Eigenvektor einer gewichteten Kovarianz) liefert, mit einem projizierten Riemannschen Gradienten-Fallback auf der Einheitssphäre, der unter milden Annahmen monotonen Abstieg gewährleistet. Distanzen werden über Amplituden-Codierung und SWAP-Tests geschätzt. Grover/Dürr–Høyer-Minimumsuche und QRAM werden als theoretische Upgrades behandelt.

Auf dem 64-dimensionalen *Digits*-Datensatz (fixe Hyperparameter) erreicht der Prototyp eine Genauigkeit, die mit einer klassischen PQ-Baseline vergleichbar ist. Bei 300 Trainingsbeispielen erzielen beide Pipelines 95 % Genauigkeit, und die normalisierten Clustering-Verluste bleiben stabil. Einschränkungen betreffen vor allem die Laufzeit sowie das Fehlen realer Quantenhardware/-speicher und dass die Zentroid-Aktualisierung aufwändiger als ein euklidischer Mittelwert ist. Nach bestem Wissen ist dies die erste durchgängige Formulierung von PQ im Rahmen der Quantuminformation. Die Ergebnisse belegen die prinzipielle Umsetzbarkeit und skizzieren einen Pfad zu praktischen Vorteilen, sobald QRAM, Quanten-Minimumsuche und quantenspektrale Routinen verfügbar werden. Kurzfristig bieten sich Varianzreduktion, Parallelisierung über Partitionen und Hyperparameter-Tuning zur Optimierung des Algorithmus an.

Der source code bezüglich der Implementierung kann unter dem folgenden Link abgerufen werden: https://github.com/ChristianJesinghaus/Hybrid-Quantum-Classical-Product-Quantization-for-Vector-Databases.

CONTENTS

1 INTRODUCTION

Modern vector databases and retrieval systems often use *approximate nearest neighbour* (ANN) search to meet latency and memory targets at scale. *Product Quantization* (PQ) is an important concept in this realm. By splitting a D-dimensional vector into m orthogonal subspaces and quantizing each part, PQ achieves strong compression while preserving neighbourhood structure for high-recall search [JDS11]. However, PQ does not remove two dominant costs: (i) evaluating many distances at indexing and query time and (ii) training subspace codebooks (typically with k-means). Against these specific bottlenecks, quantum devices offer two primitives that map directly to them. On the one hand efficient fidelity estimation between amplitude-encoded states via the SWAP test for distance calculations and quadratic savings for minimum-finding via Grover/Dürr–Høyer search [NC10; Joz94; Gro96; DH99]. In theory (assuming fast state preparation) a per-partition inner product of length d_p can be estimated in $O\big(\log d_p\big)$, instead of $O(d_p)$ classically [GLM08]. Coupled with Dürr–Høyer over c sub-centroids, the assignment step drops from $O(c\, d_p)$ to $O\big(\sqrt{c}\, \log d_p\big)$, square-root in candidates and polylogarithmic in dimension(up to state-preparation overhead).
To the best of the authors knowledge, Product Quantization has not previously been realised as a quantum algorithm. This motivates the central question of this thesis: Is it possible to implement PQ in such a way, that key parts could run on a quantum machine and if yes, does that yield a theoretical speedup?

In order to use the mentioned quantum algorithms for PQ, while staying coherent with its architecture, a key modeling choice of this thesis is to use the fidelity's negative logarithm as a distance measure:

$$d_{\ln}(\psi, \phi) := -\ln\big(|\langle\psi \mid \phi\rangle|^2 + \varepsilon\big). \tag{1.1}$$

For pure states, fidelity is multiplicative across tensor factors, but taking the negative logarithm turns it into a sum. This additivity matches PQ's "sum of per-partition distances" and preserves its efficient lookup-and-accumulate structure, while the ε-shift stabilizes cases with tiny overlaps. Eventhough there is a line of quantum k-means work - e.g. SWAP-test clustering and Grover-medoid updates [ABG06; ABG07], the "quantum Lloyd" assignment scheme [LMR13], and *q-means* [Ker+19] - they can not be used for this thesis. As they typically optimize Euclidean within-cluster variance or use overlap only for assignments while recomputing centroids as Euclidean means, the log-fidelity loss makes using their centroid update rules not applicable.

Building on that, a hybrid PQ–kNN pipeline was designed that swaps Euclidean distances for log-fidelity. With this new loss a novel rule for the centroids in k-means had to be constructed. For

that an Ansatz was taken, which uses a majorise–minimise surrogate yielding a Rayleigh–Ritz candidate and if it fails to reduce the true loss, a projected Riemannian gradient step with backtracking on the unit sphere is performed. This ensures monotone descent and blockwise stationarity under mild assumptions (see Appendix 6.1) and appears to be a new approach.

The new implementation of k-means within PQ was tested conceptually on the 64-D digits Scikit Dataset with fixed parameters against a classical implementation of PQ. As shown in Chapter 4 it performs comparably to the classical variant. An interesting insight do the clustering specific metrics give. They show that a deeper look into the finetuning of hyperparameters needs to be taken, as noise from the statistical nature of the SWAP-Tests has to be compensated for. All in all only 8 trained models each were compared and therefore a rigorous numerical testing is still pending.

A critical limitation though is the centroid update. Forming a weighted covariance and extracting its leading eigenvector costs $O(|C_j|\, d_p^2 + d_p^3)$ per cluster, versus $O(|C_j|\, d_p)$ for the Euclidean mean. A fully quantum variant would require deep circuits and significant overhead. On current NISQ hardware, noise, data access (e.g., QRAM) may dominate.

Looking ahead, once powerful enough quantum hardware exists, PQ itself may save quantum resources and an efficient quantum implementation may increase PQs training/query speed. In the interim, the in this thesis realized system uses simulated SWAP tests for distances and classical control/updates. QRAM and quantum minimum-finding are treated as theoretical upgrades. With that it offers a first formulations and prototype for a PQ pipeline that lives within Quantuminformation, giving a possible starting point for future Quantum implementations of PQ. For that Chapter 2.2 hands an Introduction to the mathematical and quantum-information background. Chapter 3 presents the system architecture and complexity comparison. The Analysis chapter 4 reports empirical results and diagnostics and Chapter 5 concludes and outlines future work.

The source code concerning the implementations can be accessed at https://github.com/ChristianJesinghaus/Hybrid-Quantum-Classical-Product-Quantization-for-Vector-Databases.

2 FUNDAMENTALS

2.1 Quantum Mechanical Basics

This Section introduces the fundamental concepts of quantum mechanics that are necessary to understand quantum-computing-specific algorithms. Unlike in classical computer science, where the smallest unit of information is the bit (0 or 1), quantum computing operates with qubits, which can exist in superposition states. It will first be explained what a qubit is and how superposition and measurement work. Basic quantum gates for one and two qubits are introduced and it is explored what it means for a set of gates to be universal — i.e., capable of implementing any quantum algorithm. Further fundamental mathematical concepts on which this Thesis and Section build on are introduced and explained in the following Section 2.2.

2.1.1 Qubit, Superposition, Measurement

Before delving into qubits, we introduce the Dirac **bra–ket notation** for quantum states. The bra–ket symbols $|\cdot\rangle$ (ket) and $\langle\cdot|$ (bra) were introduced by Dirac and are ubiquitous in quantum mechanics [Dir58]. In this notation, a quantum state vector $|\psi\rangle$ is denoted as a "ket" (column vector), and its Hermitian adjoint $\langle\psi|$ is the corresponding "bra" (row vector), i.e. the complex-conjugate transpose of the ket [Dir58]. For example, let

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}, \qquad |\phi\rangle = \begin{pmatrix} \gamma \\ \delta \end{pmatrix}. \tag{2.1}$$

Then

$$\langle\psi| = (\overline{\alpha}\ \overline{\beta}), \qquad \langle\phi|\psi\rangle = \overline{\gamma}\,\alpha + \overline{\delta}\,\beta, \tag{2.2}$$

which is the standard complex inner product in the two-dimensional state space $\mathbb{C}^2$. The two basis states of this space are written as $|0\rangle$ and $|1\rangle$, which form an orthonormal **computational basis** for a single qubit [NC10]. A **qubit** (short for *quantum bit*) is the fundamental unit of information in quantum computing, analogous to the bit in classical computing. Mathematically, a single qubit corresponds to a unit vector in the two-dimensional complex Hilbert space $\mathbb{C}^2$ [NC10]. In this space, one typically chooses an orthonormal basis — most commonly the computational basis $\{|0\rangle, |1\rangle\}$ defined above [NC10]. A general qubit state $|\psi\rangle$ is then a linear combination (superposition) of the basis vectors:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \tag{2.3}$$

where $\alpha, \beta \in \mathbb{C}$ are called *amplitudes* and must satisfy the normalization condition $|\alpha|^2 + |\beta|^2 = 1$ [NC10]. This condition ensures that the total probability of outcomes is 1 when a measurement

is performed. Expression (2.3) illustrates the concept of **superposition**: the qubit is, in general, simultaneously in both basis states $|0\rangle$ and $|1\rangle$ (with weights α and β) until a measurement occurs [NC10]. α and β can be complex numbers whose relative phases carry information with no classical analogue [NC10].
Although the relative phase of α and β is physically meaningful, superpositions cannot be observed directly. When a qubit is measured, the state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ collapses to one of the basis states — often referred to as a "collapse of the wave function" [NC10]. According to the quantum postulate of projective measurement, the outcome will be $|0\rangle$ with probability $|\alpha|^2$ and $|1\rangle$ with probability $|\beta|^2$. These probabilities are given by the squared magnitudes of the amplitudes (the Born rule) [NC10]. After the measurement, the qubit is definitively in the state corresponding to the outcome. The original superposition is destroyed and cannot be recovered [NC10]. This behavior fundamentally distinguishes quantum from classical information: a measurement yields a definite classical result (0 or 1) but also irreversibly disturbs the quantum state [NC10]. For example, it is impossible to copy an arbitrary unknown quantum state without disturbing it — this is the **No-Cloning Theorem** [NC10].

The power of superposition lies in the fact that a set of qubits spans a much larger state space than the same number of classical bits. While n classical bits can represent only one of 2^n possible n-bit values at any given time, an n-qubit system can exist in a superposition of all 2^n basis states simultaneously:

$$|\Psi\rangle = \sum_{x\in\{0,1\}^n} \alpha_x \, |x_1 x_2 \dots x_n\rangle, \qquad \text{with } \sum_x |\alpha_x|^2 = 1\,. \tag{2.4}$$

This exponential growth of the state space is the foundation for *quantum parallelism*. A quantum computer can, in principle, evaluate a function on many inputs at once by being in a superposition of those inputs [NC10]. Certain quantum algorithms can exploit this to solve problems faster than classical algorithms (most famously, Shor's factoring algorithm) [Sho97]. In Shor's case, the algorithm can factor large integers in polynomial time, whereas the best known classical methods are super-polynomial [Sho97]. However, the final result of a quantum computation must be obtained via measurement, which yields only a single outcome. This means one must interfere in an intelligent manner the amplitudes so that the correct answer is obtained with high probability, while other outcomes cancel out. Such amplitude steering techniques (e.g. Grover's search algorithm) are known as *amplitude amplification* [Gro96].
Besides the computational basis $\{|0\rangle, |1\rangle\}$—also called the Z-basis, since $|0\rangle$ and $|1\rangle$ are eigenstates of the Pauli-Z operator ($Z|0\rangle = |0\rangle$, $Z|1\rangle = -|1\rangle$)— a qubit can also be prepared or measured in other bases. An important example is the Hadamard or X-basis $\{|+\rangle, |-\rangle\}$, defined by

$$|+\rangle := \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \qquad |-\rangle := \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\,. \tag{2.5}$$

These are eigenstates of the Pauli-X operator. A measurement in the computational (Z-)basis yields

$$\Pr(0\,|\,+) = |\langle 0|+\rangle|^2 = \left|\tfrac{1}{\sqrt{2}}\right|^2 = \tfrac{1}{2}, \qquad \Pr(1\,|\,+) = \tfrac{1}{2}, \tag{2.6}$$

and similarly for $|-\rangle$:

$$\Pr(0\,|\,-) = |\langle 0|-\rangle|^2 = \left|\tfrac{1}{\sqrt{2}}\right|^2 = \tfrac{1}{2}, \qquad \Pr(1\,|\,-) = \tfrac{1}{2}. \tag{2.7}$$

Similarly, for the state $|0\rangle$ we have

$$\Pr(0\mid Z) = 1,\ \Pr(1\mid Z) = 0, \qquad \text{but} \qquad \Pr(+\mid X) = \Pr(-\mid X) = \tfrac{1}{2}. \tag{2.8}$$

Thus, the choice of measurement basis can significantly affect the outcome distribution even though the physical state of the qubit is the same. This has no classical counterpart and is central to quantum cryptography protocols (e.g., the BB84 quantum key distribution scheme) [BB84].

2.1.2 Single- and Two-Qubit Gates, Universal Gate Sets

Quantum computing, much like classical circuit design, works via logical operations applied to qubits, called quantum gates. A quantum gate is a reversible transformation of quantum state vectors, represented mathematically by a unitary matrix acting on the state space [NC10]. In particular, any 2×2 unitary matrix corresponds to a valid single-qubit gate [NC10]. Examples include the Pauli-X gate $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, Pauli-Z gate $Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$, the Hadamard gate $H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$, and the phase gates S and T (quarter- and eighth-turn Z-rotations) [NC10]. These gates act on a single qubit. For example, X flips the state of a qubit (analogous to a classical NOT gate): $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$ [NC10]. Z leaves $|0\rangle$ unchanged but flips the phase of $|1\rangle$: $Z|0\rangle = |0\rangle$, $Z|1\rangle = -|1\rangle$ [NC10]. The Hadamard gate creates an equal superposition of the computational basis states: $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |+\rangle$ and $H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |-\rangle$ [NC10].

In addition to single-qubit operations, multi-qubit gates are needed to achieve universal quantum computation. A set of gates is universal if any unitary operation on n qubits (equivalently, any n-qubit quantum algorithm) can be approximated to arbitrary accuracy using a finite sequence of those gates [NC10]. A key two-qubit gate for universality is the controlled-NOT (CNOT) gate, which acts on a pair of qubits: it flips the second qubit (target) iff the first qubit (control) is in state $|1\rangle$, and otherwise does nothing [NC10]. In the computational basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$, CNOT maps $|a, b\rangle$ to $|a, a \oplus b\rangle$ (where $\oplus$ is XOR) [NC10]. It can be shown that the CNOT gate together with a sufficient set of single-qubit gates (for instance, all phase rotations) forms a universal gate family [NC10]. Commonly used universal gate sets include arbitrary single-qubit rotations (e.g. R_x, R_y, R_z) along with the CNOT gate [NC10]. This means that any quantum computation can be decomposed into a circuit using only those gates. Universality is a crucial concept because it ensures that a small primitive set of quantum gates can be used to construct any algorithm a quantum computer might execute.

Multi-Qubit States and Tensor Products

For composite systems, the joint state space is the tensor product of the single-qubit spaces. If $|\psi\rangle, |\phi\rangle \in \mathbb{C}^2$ are single-qubit states, their joint (two-qubit) state is $|\psi\rangle \otimes |\phi\rangle \in \mathbb{C}^2 \otimes \mathbb{C}^2 \cong \mathbb{C}^4$. Written in the computational basis,

$$|a\rangle \otimes |b\rangle \equiv |ab\rangle, \qquad \text{with basis } \{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}. \tag{2.9}$$

More generally, an n-qubit pure state is a unit vector in $(\mathbb{C}^2)^{\otimes n}$ and can be expanded as

$$|\Psi\rangle = \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle, \qquad \text{where } |x\rangle \equiv |x_1\rangle \otimes \cdots \otimes |x_n\rangle, \ \sum_x |\alpha_x|^2 = 1. \tag{2.10}$$

Single-qubit gates lift to the n-qubit space by tensoring with identities. Applying a unitary U to the j-th qubit means

$$U^{(j)} = I^{\otimes(j-1)} \otimes U \otimes I^{\otimes(n-j)}. \tag{2.11}$$

Two-qubit gates act nontrivially on two tensor factors (e.g., CNOT on qubits j and k) and as the identity on the others. Any n-qubit operation is represented by a $2^n \times 2^n$ unitary matrix acting on the 2^n-dimensional state space [NC10]. The tensor product structure also clarifies entanglement. A state $|\Psi\rangle$ is called *separable* if it can be written as a product $|\psi_1\rangle \otimes \cdots \otimes |\psi_n\rangle$. Otherwise it is

entangled. For example,

$$(H \otimes I)\,|00\rangle \;=\; \tfrac{1}{\sqrt{2}}\big(|00\rangle + |10\rangle\big) \xrightarrow{\text{CNOT}} \tfrac{1}{\sqrt{2}}\big(|00\rangle + |11\rangle\big), \tag{2.12}$$

which is a Bell state and cannot be factorized as $|\psi\rangle \otimes |\phi\rangle$. Measurements on subsystems follow Born's rule with tensor-product projectors. For the Bell state above, measuring the first qubit in the Z-basis yields 0 with probability $1/2$ (post-measurement state $|00\rangle$) or 1 with probability $1/2$ (post-measurement state $|11\rangle$), revealing perfect correlations [NC10].

2.2 Mathematical Foundations

This Section assembles the mathematical toolkit needed for the log-fidelity quantum k-means algorithm and its product-quantisation variant. We recall finite-dimensional Hilbert spaces and then compare classical and quantum similarity measures, thereby motivating the use of the logarithm of the fidelity in product quantisation. Section 2.2.3 summarises the majorise–minimise (MM) principle, and Section 2.2.4 discusses optimisation on the unit sphere. A sketch of a convergence proof is provided in Appendix 6.1. For readers unfamiliar with Dirac's notation, Section 2.1.1 gives a brief introduction.

2.2.1 Hilbert Spaces

Quantum information lives in complex inner-product spaces. The state space of an n-qubit register is $\mathscr{H}_n \cong \mathbb{C}^{2^n}$. A complex vector space V equipped with an inner product $\langle \cdot \mid \cdot \rangle : V \times V \to \mathbb{C}$ that is

i conjugate symmetric,

ii linear in its left argument, and

iii positive definite

is a complex inner-product spaces. These three conditions ensure that the inner product behaves as expected. Conjugate symmetry implies that swapping the arguments only introduces complex conjugation, linearity in the first argument allows superpositions of states to be treated linearly, and positive definiteness guarantees that the norm derived from the inner product is non-negative and zero only for the zero vector. Finite-dimensional complex vector spaces with these properties are Hilbert spaces [Wer18], which form the state space of quantum information for this thesis.

Implications

In classical mechanics and in standard vector data analysis, composite systems are formed by a Cartesian product, and familiar distances (e.g., squared Euclidean) add across coordinates or blocks. Quantum systems, by contrast, compose by the tensor product of Hilbert spaces. Phases make the inner product the primary geometric object, and the induced similarity-fidelity-multiplies across independent subsystems.

Further key notions used throughout the thesis are:

- **Norm:** $\|v\|$ is the Euclidean length of v.
- **Orthogonality:** $u \perp v$ iff $\langle u \mid v \rangle = 0$.

- **Unit vector:** v with $\|v\| = 1$.
- **Euclidean distance:** $d_{\mathrm{E}}(u, v) = \|u - v\|$.

For normalised $u, v \in \mathbb{C}^d$ one has $0 \leq |\langle u \mid v \rangle| \leq 1$. Extremes such as 1 and 0 correspond to collinear (physically identical up to a global phase) and orthogonal (distinguishable) states [NC10].

2.2.2 Similarity Measures: Euclidean vs. (Log-)Fidelity

Classical setting.

In ordinary k-Means the objective is the sum of squared Euclidean distances to the current centroids d_{E} is additive across (orthogonal) coordinate axes and therefore decomposes directly in Product Quantisation pipelines[Llo82; JDS11].

Why Euclidean distance fails for quantum states.

Because the Euclidean norm depends on the global phase, two wavefunctions that represent the same physical state - such as $|\phi\rangle = \mathrm{e}^{\mathrm{i}\alpha}|\psi\rangle$ - can be separated by the maximal distance $\|\psi - \phi\| = 2$. In practice this phase-sensitivity means that physically identical states may appear far apart under the Euclidean metric, making the distances unstable and unsuitable for clustering. Fidelity, which is invariant under global phases, avoids this problem.

2.2.2.1 Fidelity

For density operators ρ, σ the (Uhlmann-)fidelity is

$$F(\rho, \sigma) = \left(\mathrm{Tr}\sqrt{\sqrt{\rho}\,\sigma\sqrt{\rho}}\right)^2 \in [0, 1] \tag{2.13}$$

and reduces to $F(\psi, \phi) = |\langle\psi\rangle\phi|^2$ for pure states [Joz94]. Fidelity is multiplicative for tensor products: $F(\psi_1 \otimes \psi_2,\, \phi_1 \otimes \phi_2) = F(\psi_1, \phi_1)\, F(\psi_2, \phi_2)$. This property introduces a key problem that was faced during this thesis, because multiplicativity is unwanted for PQ as it splits the state space into n sub-vector spaces [JDS11] and would turn the global loss into a product of n factors. This product though changes the nature of PQ fundamentally (see Section 2.3.2) and has resulted in much worse classification accuracies. This issue is adressed in the following sections by introducing a Workaround which renders the multiplicative fidelity additive and a new method for the centroid update in k-means, which works with this new loss function.

2.2.2.2 Log-Fidelity and Additivity

As the Fidelity measurement is the Go-to and well proven distance calculation on quantum machines, it would be beneficial to stay with it, while also having an additive measure suiting PQ.
We therefore work with

$$d_{\ln}(\psi, \phi) := -\ln F(\psi, \phi), \qquad d_{\ln,\varepsilon}(\psi, \phi) := -\ln\Big(F(\psi, \phi) + \varepsilon\Big), \tag{2.14}$$

where $\varepsilon > 0$ is a machine-precision safety margin that bounds the loss when $F \approx 0$. Two crucial properties can be derived on the spot:

1. **Additivity.** Because $\ln(ab) = \ln a + \ln b$,

$$d_{\ln}\big(\psi_1 \otimes \psi_2,\, \phi_1 \otimes \phi_2\big) = d_{\ln}(\psi_1, \phi_1) + d_{\ln}(\psi_2, \phi_2), \tag{2.15}$$

exactly the behaviour needed by Product Quantisation where we accumulate the sub-losses of n orthogonal subspaces.

2. **Strict convexity on** $(0,1]$**.** The map $x \mapsto -\ln(x+\varepsilon)$ is strictly convex and giving with that an important property if one wants to proof k-means convergence.

Had we kept the raw fidelity, the global loss would be $\prod_{p=1}^{n} F_p$, whose optimisation is a lot more complicated and error prone for PQ. Log-fidelity linearises this product and stabilises the optimisation. One might ask why it works to simply use the negative logarithm of the fidelity as a distance measure. Intuitively, $-\ln\big(F+\varepsilon\big)$ is phase-invariant and the convexity yields a smooth objective and enables the in Section 2.2.5 established stable Rayleigh–Ritz/gradient centroid update with monotone descent. Although $-\ln\big(F+\varepsilon\big)$ is not a true metric (it fails the triangle inequality), clustering does not require metricity — only a well-behaved, optimizable dissimilarity — so these additivity and convexity properties are exactly what the PQ pipeline needs.

Relation to the Bhattacharyya distance.

For probability vectors p, q and amplitude encoding $|p\rangle = \sum_i \sqrt{p_i}\,|i\rangle$, $|q\rangle = \sum_i \sqrt{q_i}\,|i\rangle$, the (pure-state) fidelity satisfies $F(|p\rangle,|q\rangle) = \big(\sum_i \sqrt{p_i q_i}\big)^2$. Hence,

$$-\tfrac{1}{2}\ln F(|p\rangle,|q\rangle) \;=\; -\ln\Big(\sum_i \sqrt{p_i q_i}\Big) \;=\; D_{\mathrm{Bhat}}(p,q), \tag{2.16}$$

i.e. the negative log-fidelity coincides (up to a factor) with the *Bhattacharyya distance* used classically for histogram and distribution clustering [Bha43]. This connection also explains the additivity of $-\ln F$ and supports the loss choice in light of prior Bhattacharyya k-means methods [NBS10]. Section 2.2.7 gives a more detailed look at prior work concerning the log-Fidelity and the in the next sections introduced centroid update.

2.2.2.3 Experimental Estimation

The three-qubit *swap test* returns outcome 0 with probability $p(0) = \frac{1}{2}\big(1+F(\psi,\phi)\big)$ and hence allows an unbiased estimate of F by repeated execution [Buh+01; NC10]. Many quantum machine-learning protocols, including our PQ-based implementation, are using this.

2.2.3 Majorise–Minimise (MM) principle

The following gives an insight into the well established MM-principle, which is needed for the construction of a new centroid update rule explained in more detail in 2.2.5. This centroid update rule needs to fit in with the needs of k-means in PQ (see Section 2.3 for an introduction).
Therefore let $L(x)$ be the loss to be minimised. In an MM algorithm one constructs a *surrogate* $g(x \mid x^{(t)})$ satisfying

$$g(x^{(t)} \mid x^{(t)}) = L(x^{(t)}), \qquad g(x \mid x^{(t)}) \le L(x) \quad \forall\, x, \tag{2.17}$$

minimises g in closed form, and repeats. Because $-\ln(\cdot)$ is strictly convex, a first-order Taylor expansion at c_0 yields a linear *lower* bound for the cluster loss

$$-\ln\big(|\langle\psi_i,c\rangle|^2+\varepsilon\big) \;\ge\; -\ln\big(|\langle\psi_i,c_0\rangle|^2+\varepsilon\big) \;-\; \frac{|\langle\psi_i,c\rangle|^2 - |\langle\psi_i,c_0\rangle|^2}{|\langle\psi_i,c_0\rangle|^2+\varepsilon},$$

see Section 2.2.5. After normalising these weights to sum to one, the corresponding surrogate $h(c;c_0)$ can be written as a constant minus $c^\dagger S(c_0)\,c$. Minimising h over $\|c\| = 1$ is therefore equivalent to maximising $c^\dagger S(c_0)\,c$, whose maximiser is the eigenvector associated with the *largest* eigenvalue of the weighted covariance matrix $S(c_0)$[Roc20]. This eigenvector serves as a surrogate

candidate. If it does not reduce the true loss, a Riemannian gradient step is performed in our implementation. General MM theory ensures monotonic descent and convergence to a stationary point [HL04; SBP17].

2.2.4 Optimisation on the unit sphere

Below further important mathematical concepts are introduced which are needed for section 2.2.5.

For that we treat the unit sphere $\mathbb{S}^{2d-1}$ as a Riemannian manifold with metric $\langle u, v\rangle = \mathrm{Re}(u^\dagger v)$ [AMS09]. Given a smooth function $f : \mathbb{S}^{2d-1} \to \mathbb{R}$, the Euclidean gradient $\nabla f(c)$ is projected onto the tangent space $T_c\mathbb{S}^{2d-1} = \{v \in \mathbb{C}^d : \langle v, c\rangle = 0\}$ to yield the *Riemannian gradient*

$$\mathrm{grad}\, f(c) \;=\; \big(I - c\,c^\dagger\big)\,\nabla f(c). \tag{2.18}$$

A retraction $R_c(\eta d)$ maps a tangent direction d back onto the sphere. A simple choice is $R_c(\eta d) = (c + \eta d)/\|c + \eta d\|$ [AMS09]. The Rayleigh–Ritz theorem states that for a Hermitian matrix S the maximum of $c^\dagger S c$ under $\|c\| = 1$ is the largest eigenvalue, attained by the corresponding eigenvectors[HJ12]. In our algorithm, this eigenvector provides the surrogate candidate in the centroid update.

2.2.5 Convergence of log-fidelity quantum *k*-means

This subsection hands a key part of this thesis. As already mentioned it was necessary to find a *new* rule for the centroid update due to the use of the log-fidelity loss. A detailed convergence sketch on how it works is provided in Appendix 6.1. It combines known methods to a full and to the authors knowledge novel concept which in principle should yield a solution to the problem.

The main ideas are summarised as follows:

For data states $\{\psi_i\}_{i=1}^n$ and k centroids, the loss is

$$L = \sum_{i=1}^{n} d_{\ln,\varepsilon}\big(\psi_i, \phi_{\ell(i)}\big), \tag{2.19}$$

with

$$d_{\ln}^{\varepsilon}(\psi, \phi) \;:=\; -\ln\big(|\langle\psi, \phi\rangle|^2 + \varepsilon\big). \tag{2.20}$$

The algorithm alternates between:

Assignment $\ell^{(t)}(i) = \arg\max_j \big|\langle\psi_i, \phi_j^{(t)}\rangle\big|^2$.

Centroid update For each cluster j solve

$$\phi_j^{(t+1)} = \arg\min_{\|\phi\|=1} \sum_{i:\ell^{(t)}(i)=j} d_{\ln,\varepsilon}\big(\psi_i, \phi\big). \tag{2.21}$$

Using the convexity of $-\ln$ we form a linear lower bound with weights $\hat{w}_i = \big(|\langle\psi_i, \phi_j^{(t)}\rangle|^2 + \varepsilon\big)^{-1}$, normalise these weights so that $\sum_i w_i = 1$, and construct the weighted covariance matrix

$$S_j^{(t)} = \sum_{i:\ell^{(t)}(i)=j} w_i\,\psi_i\psi_i^\dagger. \tag{2.22}$$

Its principal eigenvector (corresponding to the largest eigenvalue) maximises $c^\dagger S_j^{(t)} c$ and therefore minimises the surrogate. Denote this eigenvector by u. If the true loss $f_j(u)$ does not exceed $f_j\big(\phi_j^{(t)}\big)$, set $\phi_j^{(t+1)} = u$. Otherwise perform a Riemannian gradient step along the negative projected gradient with backtracking line search, $\phi_j^{(t+1)} \leftarrow R_{\phi_j^{(t)}}(-\eta \operatorname{grad} f_j)$, until the loss decreases.

Both steps can only decrease L. Since there are at most k^n assignments and the cost is bounded below, the process must stabilise after finitely many iterations, just as in Lloyd's classical proof [Llo82]. A sketch of a possible proof is provided in Appendix 6.1.

2.2.6 Open Convergence Questions in Prior Work

Many published quantum k-means variants simply replace the classical Euclidean distance by a quantum overlap or fidelity measure but retain the classical centroid update. As a result, formal descent proofs or convergence guarantees are rare. One noteworthy exception is the *q-means* algorithm by Kerenidis et al.[Ker+19]. In their work the authors study a δ-robust variant of classical k-means, called δ-k-means. There the labels are assigned based on Euclidean distances with a tolerance δ and centroids are computed as noisy Euclidean. They prove that, for well-clusterable datasets (clusters separated by a margin larger than δ), δ-k-means recovers the same clustering as the exact k-means, and the quantum q-means algorithm efficiently reproduces δ-k-means. However, this analysis relies entirely on Euclidean geometry. it does not extend to fidelity-based losses or the matrix-MM centroid update introduced in this work.

Table 2.1 summarises convergence claims of selected quantum k-means variants. For most, the metric is either the fidelity $F(\psi, \phi) = |\langle \psi, \phi \rangle|^2$ or its complement $1 - F$, but the centroid is still the Euclidean mean of assigned points. These heuristic methods report empirical convergence on small devices, yet no formal proof is given. The q-means algorithm remains the only variant with a proven guarantee in this list, and even that guarantee is limited to well-clusterable data under Euclidean distances. The present work introduces a log-fidelity loss and a weighted MM eigenvector update with a backtracking Riemannian gradient fallback for a hybrid quantum computing ANN algorithm. To our knowledge, no prior study has established monotonic descent or stationarity under such a loss and the quantum computing environment. But this should not be a claim that it was never done before. There might be publications within certain parts of Computer Science, Mathematics or Physics which we just didn't find, as this topic might come up in different forms within other research fields and topics.

Reference	Metric	Centroid rule	Convergence claim	Notes
Wang et al. 2021	$1-F$	Euclidean mean	empirically decreasing	small-scale IBM Q demo
Schäfer et al. 2022	$1-F$	VQE minimiser	none	variational hybrid ansatz
Pasqal blog 2022	F	Euclidean mean	none	blog article with simulations
Kim et al. 2023	F	Euclidean mean	empirical	recommender-system use case
Kerenidis et al. 2019	$\|\psi-\phi\|_2$	Euclidean mean	proven δ-k-means agreement for well clusterable data	quantum implementation of δ-k-means
This work	$-\ln(F+\varepsilon)$	MM eigenvector/-gradient	**sketch for a formal proof**	additive log-loss, fidelity-based

Table 2.1 – Convergence guarantees in the quantum k-means literature. Most approaches use a fidelity or overlap metric but update centroids by the classical mean; only *q-means* has a provable guarantee, and only under Euclidean assumptions.

2.2.7 Prior Contributions concerning the Log-Fidelity and centroid update

To contextualize the proposed *Log-Fidelity Quantum* k-*Means* and its use in Product Quantisation, this subsection summarises some relevant prior work and discusses original contributions of this thesis.

Distance measures and clustering frameworks.

Clustering based on overlap or fidelity measures has been explored in both classical and quantum settings. In the classical domain, [NBS10] developed a Bhattacharyya-divergence clustering algorithm - a generalisation of k-means using Bhattacharyya distance - and established convergence via the concave-convex procedure (CCCP). Earlier, [RTJ00] proposed an *optimal Bhattacharyya centroid* method for clustering Gaussian densities, deriving an eigenvector solution for the centroid update. The proposed method can be seen as a direct quantum analogue of Nielsen's Bhattacharyya-k-means and an extension of the concept by Aïmeur *et al.*[ABG07] to a fully specified algorithm with an outline for a formal convergence analysis.

Majorise–Minimise (MM) optimisation.

The surrogate-minorant approach for the centroid update is rooted in the majorise–minimise (MM) framework. [HL04] provide a tutorial on MM algorithms, while [YR03] formulate the related CCCP method. Both guarantee monotonic improvement by iteratively optimising a surrogate function. The here proposed algorithm applies this principle to the $-\log$ fidelity loss, producing a linearised surrogate whose minimiser has a closed-form solution via an eigenvalue problem.

Eigenvector-based centroid updates.

The Rayleigh–Ritz step in the centroid update has precedents in classical clustering. [RTJ00] derived such an eigenvector solution for Bhattacharyya-optimal Gaussian centroids, and spherical k-means similarly uses normalised means/eigenvectors as centers on the unit sphere. Our update can be regarded as a quantum generalisation of these methods to the log-fidelity loss.

Riemannian optimisation.

The fallback *Riemannian gradient* step in the proposed algorithm is based on the literature on optimisation on manifolds. [AMS08] describes projecting gradients to tangent spaces, using retractions, and performing line searches on Riemannian manifolds such as the sphere. In this setting, that guarantees a feasible descent direction whenever the surrogate update is rejected, adding robustness to the iteration.

Original contributions.

While each of the above components has precedents, their combination in a unified algorithm for clustering quantum states in an Product Quantisation algorithm is, to the best of the authors knowledge, novel. Original aspects include:

- The integration of ε-smoothed $-\log$ fidelity loss, MM surrogate update solved via Rayleigh–Ritz, and a safeguarded Riemannian gradient fallback into a Lloyd-type k-means algorithm for quantum states.
- A detailed convexity analysis and a convergence sketch for the resulting algorithm.
- The introduction of an Riemannian gradient step as a 'safety-net' in a clustering algorithm on the unit sphere.
- The integration of the above into the PQ framework, yielding possibly the first demonstration of a quantum ready algorithm of Product Quantisation for ANN search in Hilbert space.

2.2.8 Take-aways for the Quantum - PQ implementation

- The additive nature of $-\ln F$ allows *partition-wise* accumulation in Product Quantisation, whereas raw fidelity would introduce an intractable product of terms.
- The ε shift guarantees finite gradients even when $F \approx 0$ (orthogonal sub-vectors) and therefore prevents numerical overflow in the MM update.
- The MM-based centroid step has the same computational pattern as classical k-Means with covariance eigenvectors, because after the logarithmic transformation the update rule collapses to a standard covariance eigen-problem that is already encountered in other variants of k-Means.

2.3 Classical Clustering and Product Quantisation

Nearest-neighbor search (NNS) in high-dimensional spaces is a key operation in, among others, recommendation systems [Das+07], image retrieval [KG09], and audio or text retrieval [EZR19; Xio+20]. For a database containing n vectors of dimensionality D, an exact search costs $\mathcal{O}(nD)$

operations. This *curse of dimensionality* makes exact NNS quickly impractical [Bey+99; BBK01]. In the following section Product Quantization (PQ) and K-Means will be discussed, as PQ tries to tackle this issue and K-Means playes a major role in PQ.

2.3.1 Classical *k*-Means and *k*-Means++

Given a set $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_M\} \subset \mathbb{R}^D$ and a target number of clusters k, the k-means objective introduced by Lloyd [Llo82] minimises the within-cluster sum of squares

$$\mathcal{L}(\{\mathcal{C}_r\}, \{\boldsymbol{\mu}_r\}) = \sum_{r=1}^{k} \sum_{\mathbf{x}_i \in \mathcal{C}_r} \|\mathbf{x}_i - \boldsymbol{\mu}_r\|^2. \tag{2.23}$$

Iterative refinement proceeds by alternating

(E) Assignment: $\mathcal{C}_r = \left\{\mathbf{x}_i \mid r = \arg\min_s \|\mathbf{x}_i - \boldsymbol{\mu}_s\|^2\right\}$,

(M) Update: $\boldsymbol{\mu}_r = \dfrac{1}{|\mathcal{C}_r|} \displaystyle\sum_{\mathbf{x}_i \in \mathcal{C}_r} \mathbf{x}_i$.

Each iteration decreases $\mathcal{L}$ and converges to a local minimum.

2.3.1.1 *k*-Means++ seeding.

Random initial centers often lead Lloyd's algorithm into poor local optima. k-means++ [AV07] chooses the first center uniformly at random and each subsequent center $\boldsymbol{\mu}$ from $\mathcal{X}$ with probability proportional to $D(\mathbf{x})^2$, where $D(\mathbf{x})$ is the squared distance to the nearest existing center. This yields an $\mathcal{O}(\log k)$-approximation in expectation and is the default initialisation that is adopted for the classical partitions of PQ.

2.3.2 Product Quantization for Approximate kNN

Product Quantization (PQ) alleviates the curse of dimensionality by splitting the feature space into orthogonal sub-spaces and quantizing each sub-space independently. Every sub-quantizer q_j assigns its sub-vector u_j an integer code $c_j \in \{0, \dots, k_{\text{sub}} - 1\}$. Concatenating the m codes $(c_1, \dots, c_m)$ yields the actual PQ code, usually stored as an m-byte vector (for example 8 bytes for $m=8$, $k_{\text{sub}}=256$). When comparing distances these integer codes act as keys into pre-computed lookup tables, so only m memory accesses plus one summation are needed. Memory footprint and bandwidth drop by orders of magnitude [JDS11].

2.3.2.1 Vector Quantization—Basics

Lloyd Quantizer

A *vector quantizer* is a mapping $q : \mathbb{R}^D \to C$ onto a finite codebook $C = \{c_i\}_{i=1}^k$ [GN98; JDS11]. Data points are assigned via their *Voronoi cells*

$$V_i = \left\{x \in \mathbb{R}^D \mid \|x - c_i\| \le \|x - c_j\|,\ \forall j \ne i\right\}, \tag{2.24}$$

so each point is linked to its closest center c_i [Aur91; JDS11]. A codebook of size k requires $\lceil \log_2 k \rceil$ bits per vector. Within PQ the m sub-codes are stored as an m-byte integer vector. With $m = 8$ and $k_{\text{sub}} = 256$ this yields a natural 64-bit (8-byte) code. Larger k—or equivalently a larger

k_{sub}—generally lowers the quantisation error but inflates both training cost and memory footprint linearly in kD.
Quality is measured by the mean-squared reconstruction error (MSE)

$$\text{MSE}(q) = \mathbb{E}_x\big[\,\|x - q(x)\|^2\big], \tag{2.25}$$

which can be rewritten with cell probabilities $p_i = \Pr[x \in V_i]$ as

$$\text{MSE}(q) = \sum_{i=1}^{k} p_i \, \mathbb{E}_{x \in V_i}\big[\,\|x - c_i\|^2\big]. \tag{2.26}$$

The *Lloyd algorithm* [Llo82], as discussed above, iteratively minimises Equation (2.25) and satisfies after every iteration

$$q(x) = \arg\min_{c_i \in C} \|x - c_i\|, \tag{2.27}$$

$$c_i = \mathbb{E}\big[\,x \,\big|\, x \in V_i\big]. \tag{2.28}$$

The MSE decreases monotonically but converges only to local minima, making the result sensitive to the initialisation.

The iterations stop after convergence or after a preset maximum number of iterations without convergence. A single Lloyd iteration costs $\mathcal{O}(kD)$ per vector, because every point is compared with the k centers and the centroids are then recomputed.

2.3.2.2 Product Quantization

The following description of the PQ algorithm is based on [JDS11].

Code construction

Codes are generated in three steps:

1. Split $x \in \mathbb{R}^D$ into m equally long, *axis-aligned* sub-vectors $x = (u_1, \ldots, u_m)$, $u_j \in \mathbb{R}^{D/m}$.
2. Train one sub-quantizer q_j with k_{sub} centers per sub-space.
3. The overall code is $\big(q_1(u_1), \ldots, q_m(u_m)\big)$; the total number of reconstruction centers is

$$k_{\text{tot}} = k_{\text{sub}}^m. \tag{2.29}$$

Although k_{tot} grows exponentially, only $m \cdot k_{\text{sub}}$ centers are stored. Example: $m = 8$, $k_{\text{sub}} = 256 \Rightarrow$ 64-bit code. The resulting binary/integer code compresses the original vector strongly yet is structured so that later distance calculations can rely on lookup tables.

Distance estimation

Given the codes, the question is how to measure "closeness" without falling back to $\mathcal{O}(nD)$ work. Two proven estimators are used for this:

Symmetric distance computation (SDC)

$$\hat{d}(x, y) \;=\; \|q(x) - q(y)\|. \tag{2.30}$$

Asymmetric distance computation (ADC)

$$\tilde{d}(x, y) = \|x - q(y)\|, \tag{2.31}$$

where only the database vectors are quantised. Both variants need m table look-ups plus one summation [JDS11]. Empirical benchmarks show that, for the same code length, ADC typically achieves 5–10 percentage points higher *Recall* than SDC because the query vector remains uncompressed [JDJ17].

Error bound

Compression inevitably introduces error. For ADC one can show

$$\mathbb{E}\Big[(d(x, y) - \tilde{d}(x, y))^2\Big] \leq \text{MSE}(q), \tag{2.32}$$

which guarantees that the distance estimate never "blows up" as long as the quantiser is trained well [JDS11].

2.3.2.3 Non-exhaustive Search

Even with compact PQ codes a brute-force scan remains linear in database size. *Inverted File with Asymmetric Distance Computation* (IVFADC) [JDS11] avoids this by concentrating the search on certain region of the vector space.

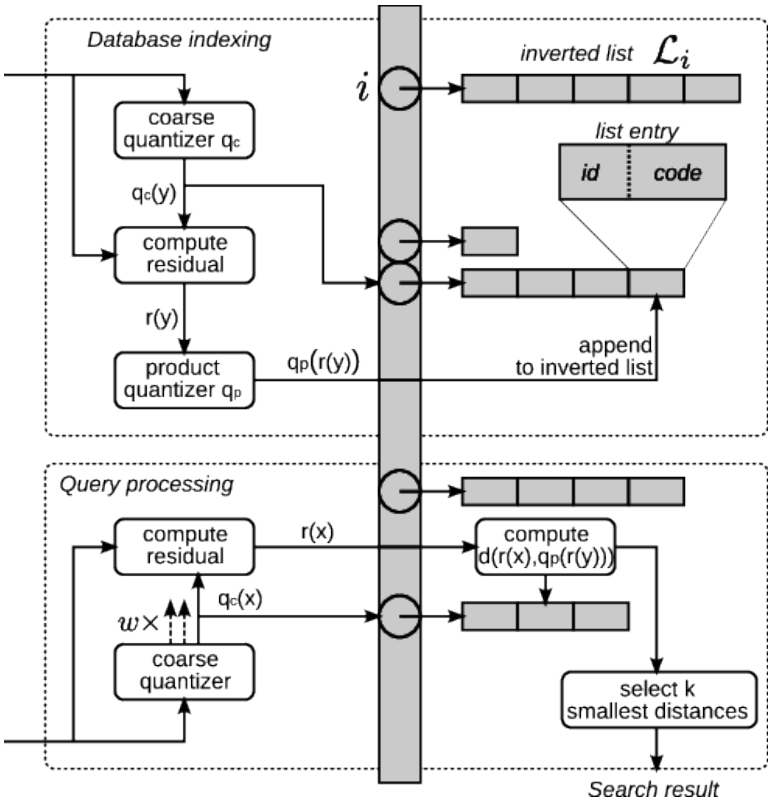


Figure 2.1 – Schematic IVFADC index (adapted from [JDS11]). The top layer shows the coarse k_0-means clusters (Voronoi cells C_i), each linked to one inverted list L_i. Residuals are PQ-encoded and stored in the list that corresponds to their nearest center. During a query only the w closest coarse cells are probed (`nprobe` $=w$); distances are evaluated by ADC lookup tables.

Combining an *inverted file* (IVF) with PQ therefore reduces both memory and computation for k-NN search:

Coarse quantizer

- A coarse k_0-means partitions $\mathbb{R}^D$ into Voronoi cells $\{C_i\}_{i=1}^{k_0}$ (Fig. 2.1).
- Each cell owns one inverted list $L_i = \{\langle \text{ID}, \text{code}\rangle\}$ that holds the vectors falling into this region.

Insertion

1. **Coarse assignment:** find the nearest center $c_i = \arg\min_{c_j} \|y - c_j\|$.
2. **Residual coding:** compute the residual $r = y - c_i$ and compress it to a PQ code $\text{code}(y) \in \{0, \ldots, k_{\text{sub}}^m - 1\}$.
3. **Posting:** store the pair $\langle \text{ID}(y), \text{code}(y)\rangle$ in list L_i.

Because only residuals are encoded, the coarse centroid already absorbs the large part of the vector norm; the remaining PQ error determines the final recall.

Query

1. **Multiple assignment:** select the $w \ll k_0$ centroids with smallest $\|x - c_i\|$.
2. **List scan:** visit only the corresponding inverted lists $L_{i_1}, \ldots, L_{i_w}$.
3. **ADC distance:** evaluate $\tilde{d}(x, y) = \|x - q(r(y))\|$ via PQ lookup tables.

Cost versus accuracy

$$\text{scanned fraction} = \frac{w}{k_0} \ (\leq 1), \tag{2.33}$$

Therefore runtime and memory accesses scale linearly with w/k_0. Larger k_0 shortens lists but raises the cost of the coarse look-up. Larger w improves recall but scans more data. [JDS11].

2.3.2.4 Selected Extensions and Recent Work

- **Optimised PQ (OPQ)** [Ge+14] – a learned rotation minimises variance between sub-spaces.
- **Additive Quantization (AQ)** [BL14] – represents vectors as a sum of several codebook vectors.
- **ScaNN (Anisotropic PQ)** [Guo+20] – score-aware quantisation plus SIMD-optimised distance computation.
- **Hierarchical PQ (HPQ)** [ABC19] – refines the search space in stages and enables pipeline search.
- **IVF-HNSW** [Pen23] – pairs an IVF coarse quantiser with an HNSW graph per list. Residuals are still compressed by PQ and yield state-of-the-art recall/latency trade-offs on billion-scale sets.

2.3.2.5 Summary and Relation to This Thesis

Product Quantization compresses high-dimensional data by orders of magnitude without heavily distorting neighbourhood structure. Combined with IVF or related indices it makes it possible to search billion-scale data efficiently. Improvements such as OPQ lower reconstruction error further and thus raise recall at unchanged memory cost.

Contribution of this work. This thesis discusses the *classical*—that is, non-rotated and graph-free—Product-Quantization algorithm as an hybrid-quantum algorithm implementation. Although extensions like OPQ offer better recall, they also increase algorithmic complexity. To isolate the acceleration potential of quantum computation and find a working implementation of the foundational algorithm, the thesis focussed solely on the plain PQ with an added k-means++ initialisation to reduce training time. Quantum Implementations of the mentioned extentions are left for future work. In the following chapters an introduction to the used quantum algorithms and techniques and their application concerning PQ is given.

2.4 Quantum Algorithms and Concepts

This chapter reviews the quantum–information theoretic building blocks that are essential for the hybrid Product-Quantisation k-NN system presented later. Section 2.4.1 motivates *quantum machine learning* (QML) and the role of quantum parallelism. Secs. 2.4.2– 2.4.5 introduce the *amplitude encoding* of classical vectors, the *Swap Test*, Grover's amplitude-amplification protocol for minimum search, and the *log-fidelity* distance.

2.4.1 Quantum machine learning and parallelism

Quantum bits (qubits) permit coherent superpositions of the form $\sum_{x\in\{0,1\}^n} \alpha_x |x\rangle$. A single unitary operation therefore acts on 2^n computational basis states "in parallel"[NC10]. Using that potential, QML research aims at algorithmic speed-ups over classical machine-learning [Bia+17]. A frequent assumption is the existence of a fast *Quantum Random Access Memory* (QRAM) that loads classical data into coherent superposition in $\mathcal{O}(\log N)$ time, where N is the data set size[GLM08]. While no scalable QRAM hardware exists yet, this assumption is also applied in the theoretical analysis of the system.

2.4.2 Amplitude encoding of classical data

Let $\mathbf{x} = (x_0, \ldots, x_{d-1})^\top \in \mathbb{R}^d$. Amplitude encoding maps $\mathbf{x}$ onto the $(\lceil \log_2 d \rceil)$-qubit state

$$|\psi_x\rangle \;=\; \frac{1}{\|\mathbf{x}\|} \sum_{i=0}^{d-1} x_i \, |i\rangle \,, \tag{2.34}$$

so that inner products of classical vectors correspond to overlaps of pure states, $\langle\psi_x|\psi_y\rangle = \frac{\mathbf{xy}}{\|\mathbf{x}\|\,\|\mathbf{y}\|}$. The helper routine `amplitude_encoding` creates the corresponding `Qiskit::QuantumCircuit` (see 'util.py'). In simulation the state-preparation cost is treated as negligible. On real hardware, it would dominate unless QRAM becomes available [Bia+17].

2.4.3 The Swap Test as a fidelity estimator

Given two n-qubit states $|\phi\rangle$ and $|\psi\rangle$ and an ancilla initialised in $|0\rangle$, the Swap Test applies a Hadamard gate, n controlled-SWAPs and a final Hadamard to the ancilla. Measuring the ancilla in the computational basis yields $P(0_{\text{anc}}) = \frac{1}{2}\big[1 + |\langle\phi|\psi\rangle|^2\big]$, therefore the (squared) overlap $F = |\langle\phi|\psi\rangle|^2$ can be estimated to precision $\mathcal{O}(1/\sqrt{\text{shots}})$[NC10]. The class `QuantumDistanceCalculator` wraps this circuit, executes it on the chosen backend and falls back to a classical dot product if execution fails.

2.4.4 Grover search and quantum minimum finding

Grover's algorithm finds a marked element in an unstructured list of size N using $\mathcal{O}(\sqrt{N})$ oracle queries [Gro96]. Dürr and Høyer showed that a variant with adaptive thresholds returns the global minimum of an unsorted array in expected time $\mathcal{O}(\sqrt{N}\log N)$ [DH99]. Module `quantum_knn_search.py` contains place-holder circuits (`create_grover_oracle`) for such oracles. The empirical evaluation still relies on classical sorting, but a code base is set up for future work on this.

2.4.5 Log-fidelity distance

For pure states the (Uhlmann) fidelity is $F(|\psi\rangle, |\phi\rangle) = |\langle\psi|\phi\rangle|^2 \in [0,1]$ [Joz94]. Because fidelity is multiplicative under tensor products, $F\big(|\psi_1\rangle \otimes |\psi_2\rangle, |\phi_1\rangle \otimes |\phi_2\rangle\big) = F(\psi_1,\phi_1)F(\psi_2,\phi_2)$, the negative logarithm,

$$d_{\ln\text{F}}(\psi,\phi) \;=\; -\ln\big(F(\psi,\phi)+\epsilon\big), \quad \epsilon > 0, \tag{2.35}$$

is additive. That property is crucial for Product-Quantisation. Splitting a D-dimensional vector into m orthogonal sub-vectors implies that the global loss $\sum_{j=1}^{m} d_{\ln\text{F}}^{(j)}$ equals $-\ln$ of the full-state fidelity. Moreover, $x \mapsto -\ln(x+\epsilon)$ is strictly convex on $(0,1]$, enabling a *majorize–minimize* (MM) update for the cluster centers in Quantum k-Means. See also the implementation in 'quantum_kmeans.py', method `_centroid_update`.

2.4.5.1 Quantum k-Means algorithms and prior work

Several quantum variants of k-means have been proposed. They share the classical iterate–assign–update structure but replace the costly parts by quantum sub-routines. A few of those are reviewed below.

Swap-test clustering [ABG06].

Aïmeur, Brassard & Gambs encode each data point as a quantum state and use the swap test to estimate the fidelity $F(\psi, \phi) = |\langle\psi|\phi\rangle|^2$. The method conceptually allows all pairwise distances to be evaluated in superposition, but it assumes that the data already reside in quantum memory.

Grover-based centroid search [ABG07].

The same authors later turned the centroid update into a *medoid*-finding task which is solved by Grover's algorithm, achieving a quadratic speed-up for that step—provided an oracle that returns the distance from a candidate point to every cluster member exists.

Quantum Lloyd's algorithm [LMR13].

Lloyd, Mohseni & Rebentrost load all M data vectors of dimension N via QRAM (amplitude encoding) and evaluate the distances to all k centroids in a single coherent query, giving an $\widetilde{\mathcal{O}}(\log N)$ cost per distance. Grover min-finding yields an overall assignment cost $\mathcal{O}(M \log(MN))$, exponentially better than the classical $\mathcal{O}(M^2 N)$ when fast state preparation is assumed.

q-Means [Ker+19].

Kerenidis *et al.* combine quantum distance estimation, amplitude-amplified nearest-centroid search, and quantum routines for centroid recomputation. The algorithm is robust to approximation errors and returns classical centroids. Runtime is poly-logarithmic in M under the QRAM assumption.

Variants and hybrids.

Wu *etal.* propose a Manhattan-distance quantum k-means with quadratic speed-up [Wu+22]. Several works map clustering to QUBO formulations for adiabatic quantum computers [KHK18; ACa21]. Hybrid schemes that off-load only the distance step to quantum hardware were explored, by Khan *et al.* on IBM devices.

Algorithm	Quantum tools	Speed-up (theory)	Key assumptions / status
Swap-test k-means [ABG06]	Fidelity via swap test	not quantified	Quantum input states required, conceptual study.
Grover-medoid [ABG07]	Grover oracle for medoid	$\sqrt{\lvert\mathcal{C}_k\rvert}$ per update	Requires distance oracle, no hardware demo.
Quantum Lloyd [LMR13]	QRAM, swap test, Grover search	exponential in N (distance) + $\sqrt{k}$	Ideal QRAM, theory only.
q-Means [Ker+19]	Full quantum pipeline	poly-log in M	Simulated, error-robust δ-clustering.
Manhattan q-k-means [Wu+22]	Amplitude enc. + Grover	quadratic	Uses L_1 norm, proof-of-concept on simulator.
QUBO annealing [KHK18; ACa21]	Quantum annealer	heuristic	Small hardware demos, no provable speed-up.

Table 2.2 – Selected quantum k-means algorithms. Speed-ups assume ideal, error-free hardware and fast state preparation.

3 ARCHITECTURE

This chapter describes the hybrid quantum Product Quantization k-NN Algorithm from theoretical and high-level point of view, down to its individual parts and implementations. The goal of the architecture is to create a system that can in theory be mostly implemented as a full quantum algorithm and then executed on real quantum hardware, while offering formal convergence guarantees. For now only few calculations are implemented as quantum circuits and testet in an ideal statevector simulation. The full or mostly quantum implementation of the presented architecture is left for future work.
In addition to a logical and physical decomposition of the software, some quality requirements and the application of the theoretical concepts introduced in the fundamentals chapter are taken into account.

The source code concerning the implementations can be accessed at https://github.com/ChristianJesinghaus/Hybrid-Quantum-Classical-Product-Quantization-for-Vector-Databases.

3.1 Theoretical approach

The theoretical Ansatz is based on three central observations. Firstly, *amplitude encoding* reduces the cost of distance evaluations: an N-dimensional vector is mapped to $log_2 N$ qubits and the inner product is estimated via SWAP-test of depth $\mathcal{O}(logN)$ instead of a classical dot-product of cost $\mathcal{O}(N)$[LMR13]. Secondly, Grover's minimum-finding algorithm can locate the closest centroid in $\widetilde{\mathcal{O}}(\sqrt{k})$ rather than k calls to an oracle[DH99]. Thirdly, the PQ decomposition itself provides a bridge between classical compression and quantum resource reduction. By splitting a D-dimensional vector into m shorter sub-vectors and representing each by a small centroid index, PQ drastically reduces memory footprint and replaces a full distance computation by the sum of m sub-distances obtained from lookup tables. In the quantum setting this translates into fewer qubits and shallower statepreparation circuits: only a sub-vector of length D/m needs to be amplitude-encoded, so the register size scales as $\log(D/m)$ rather than $\log D$. Because fidelity is multiplicative under tensorproducts, taking the negative logarithm makes it additive. The global log-fidelity loss is just the sum of the sub-losses, which perfectly matches the PQ partitioning. Each sub-vector can therefore be processed independently via the SWAP-test, and the results are accumulated classically. Together, amplitude encoding, Grover search and the additive log-fidelity yield a framework that – at least in theory – offers a significant speedup compared to classical PQ.

Scope and implemented features.

While the framework suggests amplitude encoding, swap test, Grover's minimum search, and QRAM, the implemented system follows a hybrid design. Concretely, only the *distance evaluation* is quantum-enhanced via a swap-test circuit executed on the Qiskit Aer simulator. All control flow (assignment, minimum selection, majority vote) remains classical. Grover's search and QRAM are conceptual extensions and are not part of the production path due to complexity and oracle/memory access requirements on current hardware.
Furthermore the primary objective of this thesis is to develope an algorithm, that should in principle work within the quantuminformation world and by that handing a starting point from where future work on concrete implementations can be performed.

Additivity of the log-fidelity under product quantization.

Let $x = (u_1, \ldots, u_n)$ and a centroid $c = (c_1, \ldots, c_n)$ be partitioned into n subvectors. Under the usual tensor-product model, fidelity factorizes, $F(x,c) = \prod_{p=1}^{n} F(u_p, c_p)$. Taking negative logarithms yields the additive form

$$-\ln\big(F(x,c)+\varepsilon\big) \approx \sum_{p=1}^{n} -\ln\big(F(u_p,c_p)+\varepsilon\big), \tag{3.1}$$

which aligns with PQ's lookup-sum view and motivates the log-fidelity metric with smoothing $\varepsilon > 0$ [JDS11].

Centroid update strategy.

For d_{1-F} the centroid is the normalized mean of assigned points. For $d_{\log F}$ an iteratively reweighted update is used: build a weighted covariance using $w_i = 1/(F_i + \varepsilon)$ with $F_i = |\langle x_i, c\rangle|^2$, take the (normalized) principal eigenvector as candidate, and safeguard with a backtracking step along the projected Riemannian gradient on the sphere if the candidate does not decrease the cluster loss.

3.1.0.1 Complexity: classical vs. theoretical quantum vs. real implementation

Component / Phase	Classical PQ	Theoretical Quantum-PQ (with QRAM & Grover)	Real implementation
Inner product / distance (subvector d_p vs. centroid)	Classical dot product with cost $T_{\text{dot}}(d_p) = \Theta(d_p)$ per subvector–centroid pair.	Swap test after amplitude encoding. Depth and sampling give $T_{\text{swap}} = \widetilde{\Theta}(\log d_p) + \Theta(\text{shots})$.	Swap test on Qiskit `qasm_simulator`; distances via `log_fidelity` ($-\ln(F+\varepsilon)$) or `one_minus_fidelity`; $\varepsilon > 0$ stabilises low-fidelity cases.
Assignment of one point to c centroids	Compute c distances and take `argmin`: $\Theta(c \cdot d_p)$ per point.	Grover minimum finding reduces comparisons to $\widetilde{\Theta}(\sqrt{c})$ oracle calls; each call costs T_{swap} (overall $\widetilde{\Theta}(\sqrt{c}\, T_{\text{swap}})$).	Build the distance vector to all c centroids via swap test and take a plain `argmin`: $\Theta(c \cdot T_{\text{swap}})$ (no Grover oracle used).
Centroid update per cluster j	k-means update (mean + normalisation) with $\Theta(\lvert\mathscr{C}_j\rvert\, d_p)$.	No general, proven quantum speed-up is known; updates are typically classical.	IRLS/MM with weights $w_i \propto (F_i + \varepsilon)^{-1}$, weighted covariance Σ, leading eigenvector as candidate, and a Riemannian backtracking safeguard; cost $\Theta(\lvert\mathscr{C}_j\rvert\, d_p^2 + d_p^3)$.
Training per iteration (per partition)	Assignment $\Theta(M_p c d_p)$ plus centroid updates $\sum_j \Theta(\lvert\mathscr{C}_j\rvert\, d_p)$.	Assignment $\Theta(M_p c T_{\text{swap}})$, or with Grover $\widetilde{\Theta}(M_p \sqrt{c}\, T_{\text{swap}})$; centroid updates as in the classical case.	Assignment $\Theta(M_p c T_{\text{swap}})$ plus IRLS/MM updates; early stopping when centroid shift falls below a tolerance.
Prediction: query → distance lookup	Per partition compute c distances ($\Theta(ncd_p)$) and then perform ADC-style code summation over M items ($\Theta(Mn)$).	Same structure with T_{swap} instead of d_p: $\Theta(ncT_{\text{swap}}) + \Theta(Mn)$; the additivity of $-\log F$ keeps the lookup linear.	Same as the theoretical line: $\Theta(ncT_{\text{swap}}) + \Theta(Mn)$; final decision by k-NN majority vote over the aggregated sums.

Table 3.1 – Complexity comparison of classical PQ, theoretical Quantum-PQ (under assumptions), and the real implementation. T_{swap} denotes the cost of one swap test (including shots).

Notation & assumptions.

M = #training points, M_p = #points in one partition, D = dimension, n = partitions, $d_p = D/n$ (approx.), c = centroids/partition, k = neighbours. Quantum entries assume fast state preparation / QRAM and faultless gates. Grover's minimum finding needs an oracle. The additivity of $-\log F$ enables structural optimisation advantage rather than a standalone runtime gain.

Discussion of complexity traits

Classical PQ uses per subvector euclidean distances and a mean–normalise centroid update. Theoretical-Quantum-PQ can reduce the *assignment* cost (swap-test inner products after amplitude encoding and, with Grover minimum finding, sublinear comparisons in c), but relies on strong assumptions such as fast state preparation/QRAM and efficient oracles. The current implementation adopts the quantum distance primitive in practice (swap test on Qiskit's `qasm_simulator` with smoothed log-fidelity or $1-F$), but keeps a classical control flow and does not invoke Grover. Therefore, its assignment step matches the theoretical form without the $\sqrt{c}$ gain. At the centroid step the code intentionally trades speed for robustness. Instead of a mean update it performs a safeguarded

IRLS/MM eigenvector update with a Riemannian backtracking fallback, which is more stable for the log-fidelity loss but asymptotically heavier than the classical mean.
Across all three variants, PQ's storage format (one integer code per partition) and the lookup-sum structure at prediction are identical. The additivity of the log-fidelity keeps the per-partition distance tables and the final code summation unchanged. The current code follows that pattern. It compresses into n integer sub-codes, builds per-partition query–centroid distance tables, and then sums by code indices over the database.
Pros/cons. The real implementation's advantages are (i) phase-invariant distances via fidelity, (ii) numerical stability by ε-smoothed $-\log F$, and (iii) a convergence-oriented centroid update with explicit safeguards. The costs are (a) simulator latency for swap-tests and (b) a more expensive update, which can dominate when d_p grows. The distance routine composes amplitude-encoding circuits internally, then estimates fidelity from swap-test statistics with a classical fallback, so the asymptotic assignment expression uses T_{swap} rather than d_p.

Could a quantum centroid update yield speedups later?

The IRLS/MM step reduces to (weighted) covariance formation and a leading-eigenvector computation. If, in a future setting with QRAM and block-encodings of such matrices, one could apply quantum spectral routines (e.g., phase-estimation–style eigenvector extraction or a quantum power method) to approximate the top eigenvector of the surrogate matrix, the per-cluster update might move from cubic/pseudo-quadratic costs in d_p to runtimes that depend poly-logarithmically on dimension. Realising that in practice would require efficient state preparation of the weighted point set, stable block-encodings of the surrogate, and careful error control. Until then, the centroid step remains a classical cost in both the theoretical and the implemented pipelines.

Total pipeline complexity

Per partition and training iteration, classical PQ is $\mathcal{O}(M_p c d_p)$ for assignment plus $\sum_j \mathcal{O}(|\mathcal{C}_j| d_p)$ for updates. Prediction per query is $\mathcal{O}(ncd_p) + \mathcal{O}(Mn)$ (ADC-style lookup). The real implementation replaces d_p by T_{swap} in the distance terms and uses the safeguarded IRLS/MM update, giving $\mathcal{O}(M_p c T_{\text{swap}}) + \sum_j \mathcal{O}(|\mathcal{C}_j| d_p^2 + d_p^3)$ for training and $\mathcal{O}(ncT_{\text{swap}}) + \mathcal{O}(Mn)$ for prediction (full scan over codes). The theoretical quantum-PQ matches the real implementation on distance cost and, with Grover, reduces the assignment to $\widetilde{\mathcal{O}}(M_p \sqrt{c}\, T_{\text{swap}})$. Its centroid update remains classical unless a future quantum eigenvector routine is available.

3.1.1 Algorithm overview

This subsection shows the main algorithms given as a general overview of the training, then a more specific view of the hybrid-quantum k-means clustering ansatz and finally an overview of the prediction algorithm.

General Algorithm

Input: Normalized training vectors $X \in \mathbb{R}^{M \times D}$ with labels y; number of partitions n; centroids per partition c; neighbors k; distance metric (e.g., `log_fidelity`); swap-test shots; smoothing $\varepsilon > 0$

Output: Compressed codes, learned per-partition centroids, and a prediction function

1: **Partitioning:** Let $s = \lfloor D/n \rfloor$. For $p = 1, \ldots, n-1$ set $u_p \in \mathbb{R}^s$, and let the last subvector take the remaining $D - (n-1)s$ dimensions.
2: **for** partition index $p = 1$ to n **do**
3: Train per-partition codebook via **Quantum K-Means** on $\{u_p\}$ (see Alg. 3.2); store centroids $\{c_{p,1}, \ldots, c_{p,c}\}$.
4: **end for**
5: **Compression:** Replace each subvector u_p in X by the index of its nearest centroid in $\{c_{p,j}\}_{j=1}^{c}$ to obtain integer PQ codes.
6: **Prediction:**
7: **for** each query vector q **do**
8: Split q into $(q_1, \ldots, q_n)$.
9: **for** each database item x (with stored PQ codes) **do**
10: Compute $d(q, x) \leftarrow \sum_{p=1}^{n} d\big(q_p, c_{p,\text{index}_p(x)}\big)$, where the distance uses either $1 - F$ or $-\ln(F + \varepsilon)$; if configured, estimate F via swap test.
11: **end for**
12: Select the k smallest distances and return the majority label among the corresponding y.
13: **end for**
14: **return** compressed codes, centroids, prediction function

---

Algorithm 3.1 – Shows how `QuantumProductQuantizationKNN.compress` works. Integer block partitioning. Per-partition `QuantumKMeans.fit_predict` with k-means++ and quantum distances. Writes indices to `compressed_data` and centroids to `subvector_centroids`.

Hybrid-Quantum K-Means (overview)

Input: Partition data $U = \{u_i\} \subset \mathbb{R}^d$; clusters c; metric $\in \{1 - F, -\ln(F + \varepsilon)\}$; smoothing $\varepsilon > 0$; swap-test shots
Output: Centroids $\{c_1, \ldots, c_c\}$ and assignments

1: Initialize centroids with k-means++ (under the chosen metric).
2: **repeat**
3: **Assignment:**
4: **for** each $u_i \in U$ **do**
5: Assign u_i to $j^\star = \arg\min_{j \in \{1,\ldots,c\}} d(u_i, c_j)$ (distance via swap test with fallback for F, then $1 - F$ or $-\ln(F + \varepsilon)$).
6: **end for**
7: **Update:** $C \leftarrow \text{UpdateCentroids}(U, C, \text{metric}, \varepsilon, \text{shots})$ /* see Alg. 3.3 */
8: **until** centroids have converged (e.g., $\|C^{(t)} - C^{(t-1)}\| <$ `tolerance`) or maximum iterations
9: **return** C

Algorithm 3.2 – Corresponds to `QuantumKMeans.fit(...)`: outer iteration with quantum assignment and the safeguarded centroid update.

Hybrid-Quantum K-Means: Centroid update (detail)

$C' \leftarrow C$
for $j = 1$ to c **do**
  if metric is $1 - F$ **then**
    $c'_j \leftarrow \text{Normalize}\left(\sum_{u_i \in C_j} u_i\right)$
  else
    /* IRLS candidate via reweighted covariance */
    Compute $F_i = |\langle u_i, c_j \rangle|^2$ for $u_i \in C_j$; set $\tilde{w}_i = \dfrac{1}{F_i + \varepsilon}$ and normalize w_i.
    $\Sigma = \sum_{u_i \in C_j} w_i \, u_i u_i^\top$; let m be the leading eigenvector and $c_{\text{cand}} = \dfrac{m}{\|m\|}$.
    $f(c) = \sum_{u_i \in C_j} -\log\left(|\langle u_i, c \rangle|^2 + \varepsilon\right)$
    if $f(c_{\text{cand}}) \leq f(c_j)$ **then**
      $c'_j \leftarrow c_{\text{cand}}$
      /* accept Rayleigh candidate */
    else
      /* Safeguard: backtracking along projected gradient (sphere) */
      $g \leftarrow \Sigma c_j$; $d \leftarrow g - \langle c_j, g \rangle \, c_j$
      if $\|d\| < 10^{-12}$ **then**
        $c'_j \leftarrow c_j$
        /* flat region */
      else
        $d \leftarrow d / \|d\|$, $\alpha \leftarrow 1.0$, $f_{\text{old}} \leftarrow f(c_j)$
        for $t = 1$ to 10 **do**
          /* at most 10 halvings */
          $c_{\text{try}} \leftarrow \text{Normalize}(c_j + \alpha \, d)$
          if $f(c_{\text{try}}) < f_{\text{old}}$ **then**
            $c'_j \leftarrow c_{\text{try}}$
            break
          else
            $\alpha \leftarrow \alpha / 2$
          end if
        end for
      end if
    end if
  end if
end for
return C'

Algorithm 3.3 – Shows the implementation of IRLS/Rayleigh candidate for log-fidelity with backtracking along the projected gradient (< 11 halvings) and the normalised mean update for $1 - F$. See `_centroid_update(...)`.

Prediction algorithm

Input: Query set $Q \in \mathbb{R}^{T \times D}$; learned partition size s; number of partitions n; centroids $\{C_p\}_{p=1}^{n}$ with $C_p \in \mathbb{R}^{c \times d_p}$; compressed training codes Codes $\in \{0, \ldots, c-1\}^{M \times n}$; training labels $y \in \{1, \ldots, L\}^M$; neighbors k; distance metric (`log_fidelity` or `one_minus_fidelity`); shots; smoothing ε
Output: Predicted labels $\hat{y} \in \{1, \ldots, L\}^T$

for each query $q \in Q$ **do**
  Split: For $p = 1, \ldots, n-1$ set $q_p = q[(p-1)s : ps]$, last subvector $q_n = q[(n-1)s : D]$.
  Centroid distances: For each $p = 1, \ldots, n$ compute a length-c vector $D_p[j] \leftarrow d\big(q_p, C_p[j]\big)$ via the quantum distance calculator (swap test with shots; fallback if needed).
  Approx. database distances: Initialize $d_{\text{sum}}[i] \leftarrow 0$ for $i = 1, \ldots, M$.
  for $p = 1$ **to** n **do**
    for $i = 1$ **to** M **do**
      $j \leftarrow \text{Codes}[i, p]$
      $d_{\text{sum}}[i] \leftarrow d_{\text{sum}}[i] + D_p[j]$
    end for
  end for
  Vote: Let $\mathcal{N}_k$ be indices of the k smallest values in d_{sum}.
  $\hat{y}(q) \leftarrow$ majority label among $\{y[i] : i \in \mathcal{N}_k\}$.
end for
return $(\hat{y}(q))_{q \in Q}$

Algorithm 3.4 – Shows how `QuantumProductQuantizationKNN._partition_dists()` and `_predict_one()` work.

3.2 Constraints and Assumptions

3.2.1 Technological Constraints

- **Hybrid approach:** The bulk of the processing (data preparation, partitioning, encoding) is executed classically on the CPU, whereas distance evaluations optionally employ quantum mechanical subroutines. The architecture provides fallbacks to classical operation when no quantum hardware is available or a swap test simulation fails.

- **Data characteristics:** We assume high-dimensional numerical vectors (e.g. image or audio features). PQ divides the D-dimensional vector into n equally long sub-vectors and quantises each with k centroids[JDS11].

- **Quantum simulators:** The distance calculations are executed on Qiskits *Aer qasm_simulator*. Futher quantum algorithms like Grover are currently not implemented and used. For most of the quantum speedup real QRAM would be needed [GLM08]. A simple implementation and stub for a later QRAM implementation is given, but not used.

- **Libraries:** External dependencies are limited to `numpy`, `scikit-learn`, `qiskit` and a few visualisation libraries (see `requirements.txt`).

3.2.2 Domain Assumptions

- **Correctness of log-fidelity k-means:** The `QuantumKMeans` algorithm implements the MM procedure with re-weighted eigenvectors as described in Section 2.2.3. The monotonic decrease of the loss and convergence to stationary points are assumed due to the mathematical thoughts in 6.1.
- **Subvector independence:** PQ assumes that the sub-vectors form independent subspaces. Only then is the additive decomposition of the log-fidelity meaningful[JDS11]. The algorithm therefore stores separate centroids and compression results per partition.
- **Majority vote classification:** In the kNN step the most frequent label among the k nearest vectors is chosen.

3.3 System Overview

3.3.1 System Context

Figure 3.1 shows the overall context of the hybrid Quantum-PQKNN system. User defined hyperparameters are stored in a text file and parsed by `ConfigLoader`. The raw data are then loaded and normalised using the utility functions in `normalize.py`. Training and inference are done by `QuantumProductQuantizationKNN`. During training the class invokes `QuantumKMeans` on each partition of the data to build a sub-codebook, and during prediction it calls`QuantumDistanceCalculator` to estimate log-fidelity overlaps via the SWAP test. All models and experiment metadata are saved by `ModelPersistence`. The quantum routines themselves, specifically the SWAP test, are implemented with Qiskit's *Aer* simulator inside `QuantumDistanceCalculator`. There is no separate `QuantumSimulator` wrapper, although this file exists and only consists of a QRAM stub and some simulator implementations for possible later integration. Auxiliary routines such as `amplitude_encoding` and the experimental Grover oracles reside in `util.py`.

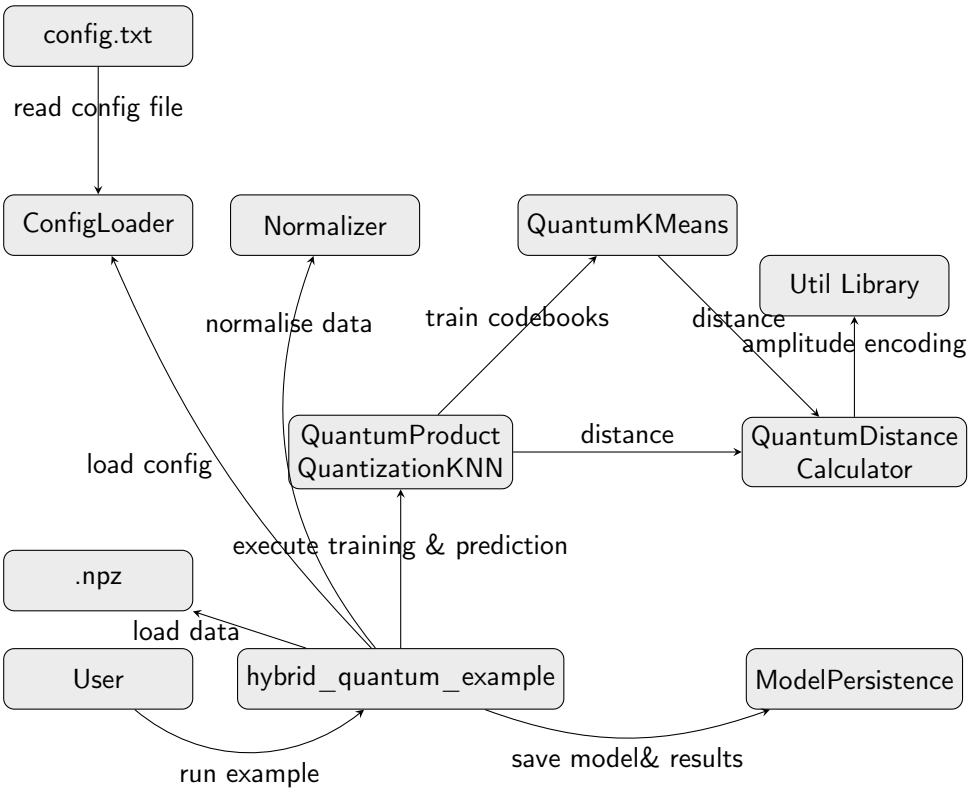


Figure 3.1 – Overview of the interaction between the main files, classes and actors.

3.3.2 High-Level Process Flow

The end-to-end workflow comprises three phases:

1. **Initialisation.** A configuration file defines the training/test sizes, number of partitions n, sub-centroids c, the chosen distance metric (e.g. `log_fidelity`) and, for the quantum path, the number of SWAP-test shots. `ConfigLoader` reads these key–value pairs, validates them and fills in default values.

2. **Training and compression.** The script `hybrid_quantum_example.py` loads the dataset (for example, from an `.npz` file), normalises it via `normalize.py`, gets the configurations from `ConfigLoader` and calls `QuantumProductQuantizationKNN.compress()`. Each D-dimensional input vector is split into n subvectors, a `QuantumKMeans` instance is trained on each subspace using the reweighted eigenvector update, and the training data are encoded as integer PQ codes stored in `self.compressed_data`. The learned centroids are kept in `self.subvector_centroids`.

3. **Prediction.** To classify a query, `QuantumProductQuantizationKNN._predict_one()` decomposes the query into n subvectors, computes the log-fidelity distance to each compressed training vector via `QuantumDistanceCalculator.quantum_distance_matrix()` (with a classical fallback if the SWAP test fails), sums the sub-distances and selects the majority label among the k smallest sums. Predictions and auxiliary results are saved by `ModelPersistence`, and the helper script `test_saved_model.py` can reload a stored model for verification.

3.4 Component View / Component Model

3.4.1 Component Model

The most important classes and functions are assigned to their responsibilities in Table 3.2. The assignment is based on the source code (see Listing 6.1 for a central example).

Component	Implementation (file/class)	Responsibility and role
Configuration management	`txt_config_loader.py` / `ConfigLoader`	Reads key–value configuration, applies defaults, validates inputs.
Configuration file	`config.txt`	Contains experiment parameters (partitions n, clusters c, neighbours k, shots, distance metric, etc.).
Data normalisation	`normalize.py` / functions	Provides various normalisation methods (L2, min-max, standard, quantum-amplitude normalisation).
Classical PQ compression	`PQKNN.py` / `ProductQuantizationKNN`	Performs classical k-means per partition.
Quantum k-Means	`quantum_kmeans.py` / `QuantumKMeans`	Implements the MM procedure with re-weighted eigenvector candidates, safeguard via Riemannian gradient and k-means++ initialisation.
Distance calculation	`quantum_distance.py` / `QuantumDistanceCalculator`	Computes log-fidelity or fidelity via swap test; provides classical fallback.
Hybrid PQ-kNN	`quantum_pqknn.py` / `QuantumProductQuantizationKNN`	Orchestrates partitioning, invokes `QuantumKMeans.compress()` per partition, maintains `compressed_data` and performs kNN predictions.
Grover/QAE utilities	`util.py`	Contains quantum state preparation, Grover oracles and other functions for possible future implementations.
Persistence	`model_persistence.py` / `ModelPersistence`	Saves and loads trained models, configurations and results; produces summaries.
Quantum simulator	`quantum_-simulator.py/QuantumSimulator`, `QRAMSimulator`	Placeholder classes for circuit execution and QRAM. They are not used in the current training/prediction workflow.
Scripts	`hybrid_quantum_example.py`, `test_saved_model.py`	Demonstrate the end-to-end workflow (training, compression, prediction, persistence, evaluation).
Data preparation	`create_digits_npz.py`	Generates NPZ files from the Digits dataset for use in experiments.

Table 3.2 – Mapping of the main components to their implementations and roles.

3.5 Runtime View

Figure 3.2 illustrates the training phase as a sequence diagram. A query scenario is given in 3.3. Both training and prediction proceed sequentially over partitions in the current implementation. No parallel execution is assumed.

Sequence diagram of the training phase

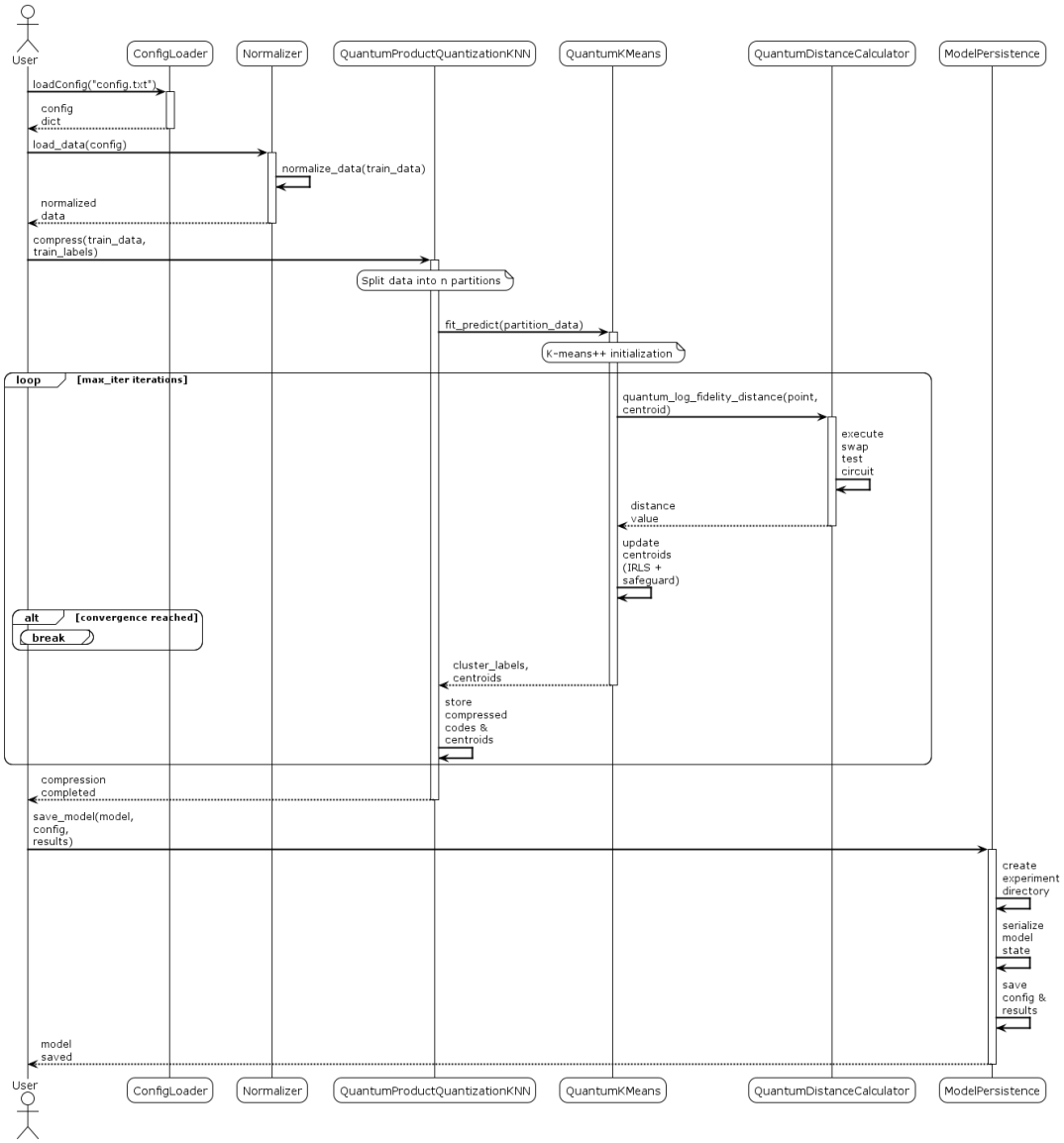


Figure 3.2 – This figure shows a sequence diagram of the training phase. For complexity reasons the hybrid_quantum_example and config files are excluded. It was generated with PlantUML.

Sequence diagram of the prediction phase

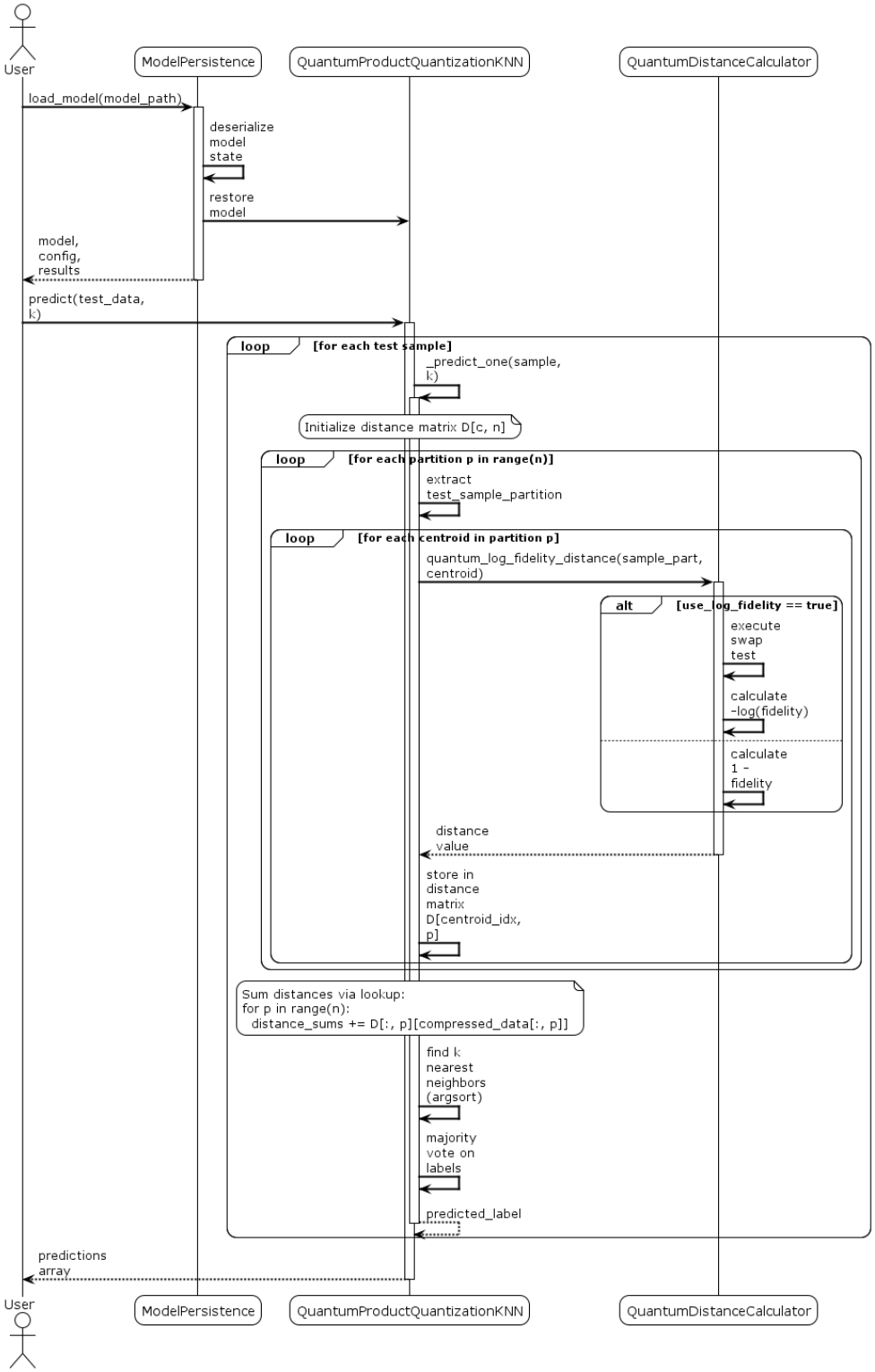


Figure 3.3 – Shown is the sequence diagram of the prediction phase. As before, the example and config file are excluded. The figure was generated with PlantUML.

3.6 Persistence Model

Each experiment is named with a timestamp and the some configuration parameters (`generate_experiment_name`). `ModelPersistence.save_model()` creates a directory `experiments/models/mode timestamp` and stores:

- **model.pkl**: The serialised model with centroids, compression data, labels and hyperparameters.
- **config.json**: The original configuration.
- **results.json**: A summary of results (e.g. accuracy).
- **quantum_info.json**: Information on the distance metric, shots, and ε.
- **compression_stats.json**: Ratio, shape of the data, number of centroids.
- **history files**: Optional JSON files per partition capturing the Quantum KMeans history (backtracking, loss, gradients).

3.7 Testing

With the test file one can load existing models and test them with either the original configs, or by varying the number of neighbours used for the prediction decision. The accuracy results can then be compared to the original results.

4 ANALYSIS

This chapter presents an analysis of the experiments carried out with the classical and hybrid-quantum Product-Quantisation k-Nearest-Neighbour (PQ-kNN) algorithms. Unlike the architectural and theoretical chapters, the focus here lies on the empirical behaviour of the algorithms. For example classification accuracy, convergence behaviour, cluster quality and how these quantities scale with the training set size are shown an discussed. Throughout this chapter all plots refer to experiments conducted on the ”Digits” dataset (64-dimensional hand-written digit images with ten classes) generated by `create_digits_npz.py`. The data was L2-normalised, split into a training set of size M and a test set of size $T = \lfloor M/5 \rfloor$, and quantised with $n = 8$ partitions and $c = 10$ centroids per partition. The number of nearest neighbours in the final k-NN stage was fixed to $k = 9$. The classical pipeline uses Euclidean distances between sub-vectors and k-means centroids, while the quantum pipeline employs the log-fidelity metric estimated via SWAP tests.

Classical PQ implementation provenance.

The classical PQ kNN component builds on the open-source implementation by Van der Donckt *et al.* (`https://github.com/jvdd/pqknn`), used under the MIT License.[1] The squared Euclidean subvector distances and scikit-learn KMeans are kept and integrated the implementation into the persistence and reporting stack for like-for-like comparisons with the hybrid-quantum pipeline.

Code of the tested implementation

The source code concerning the implementations can be accessed at https://github.com/ChristianJesinghaus/Hybrid-Quantum-Classical-Product-Quantization-for-Vector-Databases.

4.1 Experimental setup and metrics

For each run the configuration (train/test sizes M and T, number of partitions n, centroids c, neighbours k, seed, distance metric and shots for the SWAP test) and the resulting statistics were stored in JSON files by the persistence layer. With a plotting script all runs were collected into a *DataFrame* and computed to the aggregated metrics:

- **Accuracy.** The fraction of correct predictions on the test set. Given as $A = \frac{\#\text{correct classifications}}{\#\text{all classifications}}$. Classical and quantum accuracies are plotted against M in Figure 4.1.

[1] MIT License (c) 2021 Jeroen Van Der Donckt. The full license text is included among the project code accompanying this thesis.

- **Training and prediction times.** The time to learn sub-quantisers and to classify the test set. These are strongly influenced by the computational load of the host machine and are therefore reported for completeness but not used for strict comparison.
- **Iteration counts.** For the classical algorithm the average number of Lloyd iterations per partition and for the quantum algorithm the average number of majorise–minimise (MM) updates per partition are recorded. These values indicate convergence speed.
- **Cluster loss measures.** For classical runs the final within-cluster sum of squared errors (*inertia*) per partition is stored and averaged. For quantum runs the final log-fidelity objective is averaged. As both losses grow with dataset size, per-point normalisations are also computed. These give the average loss contribution per training vector and allow cluster quality to be compared across data sizes.
- **Objective value**: the log-fidelity loss after the update.
- **Accept ratio**: the fraction of centroids for which the Rayleigh–Ritz candidate is accepted. A lower accept ratio signals that the safeguarded Riemannian gradient step is needed more often.
- **Gradient norm**: the maximum Euclidean norm of the gradients across centroids. Declining gradients indicate approach to a stationary point.
- **Centroid shift**: the weighted sum of centroid movements during the update. Small shifts reflect stabilised centroids.
- **Iteration time**: wall-clock time spent on the MM step.

Stopping criteria.

Unless stated otherwise, each quantum k-means instance (per partition) stops if (i) the centroid shift falls below $\tau = 2 \cdot 10^{-3}$, (ii) a hard iteration cap is reached. Section 4.3.2 reports an ablation with a looser threshold ($\tau = 10^{-2}$).

Table 4.1 summarizes the collected Data of all runs. It contains eight classical and eight quantum runs with training sizes $M \in \{30, 60, 90, 120, 150, 200, 250, 300\}$. The detailed k-means iteration information were omitted in this table for space reasons. The training/predict Times as well, as they heavily relied on the overall machine load. But generally one can say, that while the training times for the classical models took only few seconds at maximum, the training time for the quantum model took several hours or even days at the largest model. This is mostly due to the much more complex computations in the quantum simulations and the overall algorithm. Another reason might also be that the hybrid-quantum implementation only computes sequentially and not in parallel.

While, in principle, eight trained models could be enough to indicate a trend if supported by statistical significance, the present sample does not permit such inference or adequate uncertainty quantification. The algorithm has many parameter settings and is a novel approach of k-means in the context of Product Quantization. Furthermore the influence of the quantum SWAP-Test on this clustering method is unknown. Therefore a rigorous evaluation would require many more trained models under different configurations in order to see how the different parameters affect the models

behaviours. Additionally more models should be trained and tested on substantially larger and more diverse datasets to probe scalability and robustness. Future experiments should scale M and D up and evaluate classification performances and clustering metrics. Nonetheless this Chapter uses these eight for a proof of concept and an early analysis of the convergence behaviour of the proposed algorithm. The small amount of trained models is due to the reason, that train times of the quantum model for larger M were very long on the used machine.

Algorithm	M	Accuracy	Mean iters	Loss	Loss/pt
Classical	30	0.33	2.5	0.172	0.046
Classical	60	0.58	3.8	0.433	0.058
Classical	90	0.72	5.5	0.759	0.068
Classical	120	0.83	5.9	1.011	0.067
Classical	150	0.97	6.8	1.355	0.072
Classical	200	0.93	7.3	1.759	0.070
Classical	250	0.94	9.0	2.259	0.072
Classical	300	0.95	13.8	2.737	0.073
Quantum	30	0.33	4.9	1.106	0.295
Quantum	60	0.50	6.4	3.415	0.455
Quantum	90	0.67	7.5	5.525	0.491
Quantum	120	0.83	9.8	7.066	0.471
Quantum	150	0.83	12.5	9.388	0.501
Quantum	200	0.90	10.9	12.618	0.505
Quantum	250	0.92	28.3	16.191	0.518
Quantum	300	0.95	20.1	19.131	0.510

Table 4.1 – Summary of experimental runs. Mean iters denotes `mean_kmeans_iter` for classical runs and `mean_qkm_iter` for quantum runs. Loss denotes the average inertia (classical) or average log-fidelity objective (quantum) across partitions. Loss/pt is the corresponding per-training-point normalisation. Training and prediction times were omitted from the table because they depend on machine load. Complete values are available in the JSON logs.

4.2 Classification performance

Figure 4.1 compares the classification accuracy of the classical and quantum PQ-kNN implementations as a function of the training set size M. For both algorithms the accuracy rises monotonically with M. However the quantum method consistently underperforms its classical counterpart on smaller data sets. Starting at $M = 30$ both algorithms achieve an accuracy of 0.33, reflecting the difficulty of learning from just a few digits. As the dataset grows, the quantum accuracy lags behind by roughly 5–10 percentage points until $M = 150$. Beyond $M = 200$ the gap narrows, and at $M = 300$ both variants reach an accuracy of 0.95. This convergence together with the overall trend of both functions could suggests that the quantum distance measure does not degrade classification when sufficient training data are available.

An interesting Datapoint lies at M = 150. Here both algorithms seem to diverge and the prior and posterior trend of similar functional behaviour and accuracies seem to break. This could be due to characteristics of the test/train dataset with which the quantum algorithm performs significantly worse and interestingly the classical one better on. If this behaviour appears regulary in future analysis on more models, one might investigate the digits datasets on which these models were trained and find out if and what characterics they have, which lead to this difference.

As mentioned previously this accuracy comparison serves solely as a proof of concept, which should show, that in small datasets the proposed algorithm can perform approximately as good as the classical variant. Further, more detailed analysis with more Datapoints should be performed in the future.

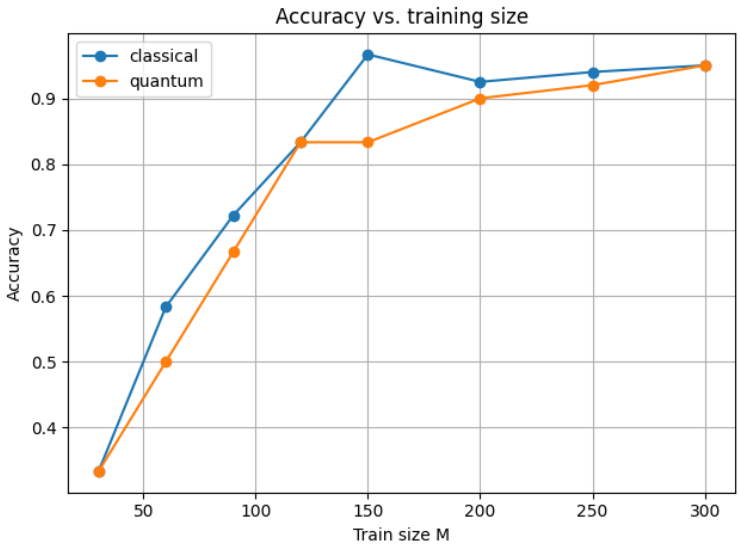


Figure 4.1 – Accuracy vs. training set size M for classical and quantum PQ-kNN (values from Table 4.1). The classical pipeline consistently achieves higher accuracy on small data sets, but the quantum pipeline converges to the same level for the largest run ($M = 300$).

4.2.1 Confusion matrix analysis

To complement the aggregated accuracy scores, confusion matrices for selected runs were also computed using the saved models and the original Digits dataset. Figure 4.2 depicts the normalised confusion matrices for the largest classical and quantum runs ($M = 300$). Each cell at row i, column j shows the fraction of test examples of true class i that were assigned to predicted class j. Perfect classification corresponds to the identity matrix. As one can clearly see for $M = 300$ both models behave very similar. The missclassifications for $True = 1, 3, 9$ lie in the same classes each time. It is reasonable to assume, that these correspond to exactly the same test images for both models. This supports the previously stated hypothesis, that both models converge with larger amounts of training data.

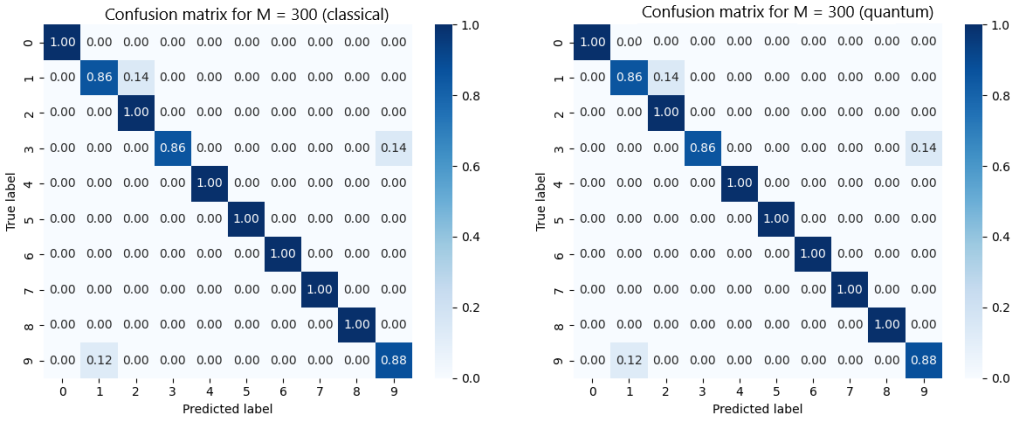


Figure 4.2 – Normalised confusion matrices for the classical (left) and quantum (right) PQ-kNN on the $M = 300$ run. Rows correspond to true digit classes, columns to predicted classes. Colour indicates the proportion of examples. A perfect classifier would show intensity only on the diagonal.

Figure 4.3 on the other hand shows the same plot for the $M = 150$ models. These are the models, where there is a divergence in the accuracies as observed in Figure 4.1. One can directly see the most relevant distinction, at classes 1 and 2 as well as 4. In the quantum model Class 1 is falsely classified as 2 in 50 percent of the times and Class 2 as 1 in 25 percent of the times and as 7 in another 25 percent. This might be due to the similarity of these classes as handwritten digits, which is shortly shown and discussed in Figure 4.4 and Figure 4.5. The same can be said about the missclassification of Class 4 as 1. And interestingly enough the only appearing false prediction of the classical model appears in the similar way in the quantum model. Class 9 is predicted as 1 in 33 percent of the times in both models. Notably this is the exact same error appearing also in Figure 4.2, just with other percentages as the amount of test Data differs. This is another hint on very similar looking pictures in this instance.

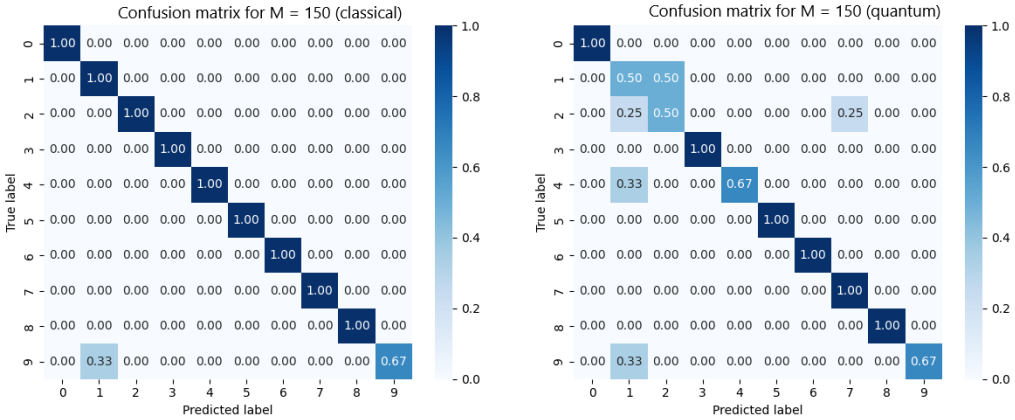


Figure 4.3 – Normalised confusion matrices for the classical (left) and quantum (right) PQ-kNN on the $M = 150$ run. Rows correspond to true digit classes, columns to predicted classes. Colour indicates the proportion of examples. A perfect classifier would show intensity only on the diagonal.

Figure 4.4, 4.5 both show the images used for training on handwritten 2's and for testing on handwritten 1's the $M = 150$ models. The test set only contains two images and therefore as derivable from the corresponding confusion matrix one of these images was falsely predicted as a two. By looking at the train set of the 2's and comparing it to the test set, one can most likely say that the left test picture was predicted as a two. Some of the train pictures look quiet similar to this test picture. And most apparend are the small test and train sizes. On small sets like these are fluctuations in the models accuracy not surprising and it rather strengthens the argument, that the proposed algorithm performs well, as it's accuracy is comparable to the classical one for most trained models.

Test images

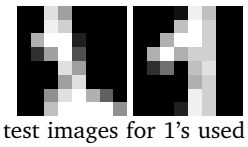


Figure 4.4 – Shown are the two test images for 1's used to test the $M = 150$ models. The pictures are taken out of the .npz and the same as those used for training and testing the $M = 150$ models.

Train images

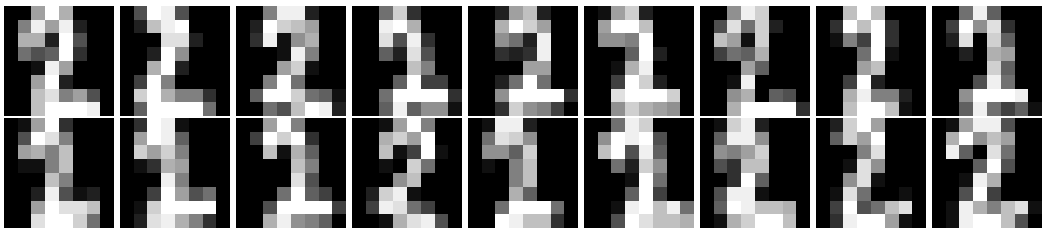


Figure 4.5 – Shown are the train images for 2's used to train the $M = 150$ models. The pictures are taken out of the .npz and the same as those used for training and testing the $M = 150$ models.

4.2.2 Classification - Summary

Both algorithms exhibit strong diagonals in the $M = 300$ models, confirming their high accuracy on the largest dataset. The off-diagonal values highlight remaining confusions. Especially with the $M = 150$ quantum-Model this is a problem in the Classes 2 and 1. Confusions happening in this model might be due to the optical similarity and with at this stage still lower model accuracy, some train/test sets lead to a divergence in the models behaviour. Nevertheless the error patterns are similar overall, and the quantum matrix approaches the classical matrix as M increases. If this trend extrapolates with more training/test data, then both models converge in their classification accuracy and the quantum model performs at least as well as the classical.

4.3 Iteration and convergence behaviour

The average number of iterations required for each sub-space quantiser to converge is plotted in Figure 4.6. For classical runs the Lloyd algorithm stabilises within 2–14 iterations, with a gentle increase as M grows. In contrast, the quantum algorithm exhibits a pronounced spike at $M = 250$, where the mean number of MM updates jumps to 28.3 before dropping again at $M = 300$. This outlier suggests that the reweighted eigenvector update and backtracking procedures occasionally need many iterations to settle when the training set size triggers a particularly challenging combination of cluster assignments. For $M = 150$ the quantum model interestingly does not show a significant change in the trend, as in the accuracy. This could be a hint, that the divergence in accuracy is not correlated to a general clustering problem. Overall, the quantum iteration counts remain within the same order of magnitude as the classical ones. If one ignores the spike at $M = 250$ both models follow the same trend, but the quantum model constantly needs more iterations.

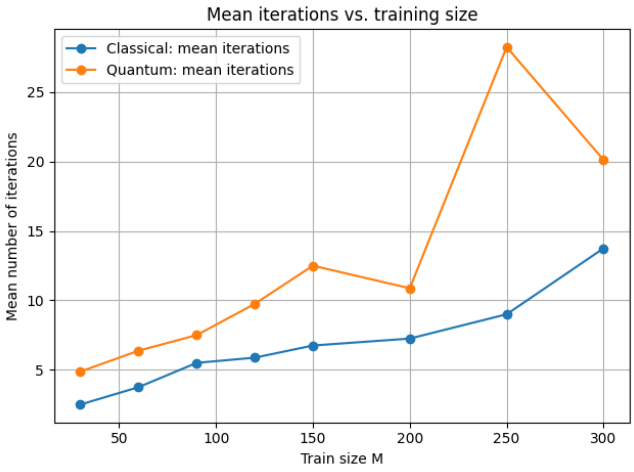


Figure 4.6 – Mean number of cluster-update iterations per partition vs. training size M. Classical runs use the Lloyd update, quantum runs use a majorise–minimise eigenvector update with backtracking. A spike at $M = 250$ indicates a harder clustering instance for the quantum method.

Another way to view convergence is through the average training time per iteration (Figure 4.7). Although absolute times are hardware and load dependent, the logarithmic scale reveals that the quantum updates are significantly slower than the classical updates. This stems from constructing and simulating swap-test circuits for each distance evaluation and the overall more complex and less parallelized implementation. By looking at Figure 4.7 one can see, that both models do not follow the same trend. The classical models train-time decreases with the amount of data and the quantum models time increases. As mentioned before the train times heavily varied with the current load of the used machine. Therefore this divergence could just be an artifact of the small Dataset, but it should be further looked at in the future.

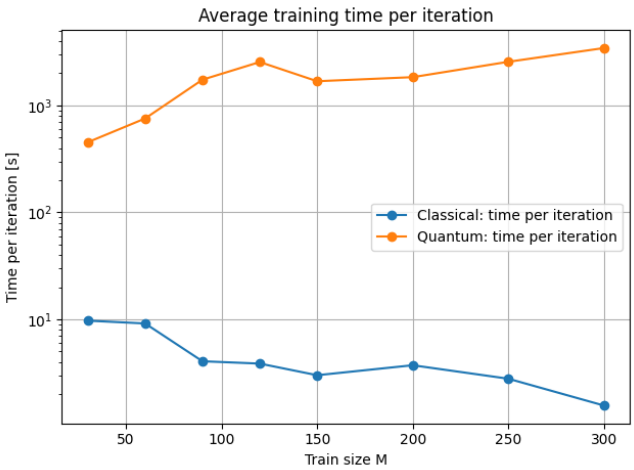


Figure 4.7 – Average training time per iteration vs. training size. Quantum updates (MM + log-fidelity) are 10–1000× slower than classical Lloyd iterations.

4.3.1 Detailed quantum iteration diagnostics

While the mean iteration counts provide a coarse measure of convergence, they do not reveal why certain runs require many more updates than others. To gain deeper insight into the quantum training dynamics the algorithm was instrumented to record several quantities at every majorise–minimise (MM) step for each partition.

The plotting utility aggregates these metrics across partitions and produces one curve per metric for each run. For example, Figure 4.9 shows the diagnostics for the $M = 250$ quantum run (the one with the mean iteration count anomaly) in comparison to the $M = 300$ model. For the $M = 250$ Model the objective drops steeply over the first twelve updates and then enters a large plateau in which the objective value oscillates around that plateau. On the first sight it looks misleadingly as if there are still large steps between plateaus, but this is due to the reason that the Figure shows an average over the partitions. In Figure 4.8 one can clearly see, that these steps always occure on those points, where the last iteration of a partition is reached. And because the objective value is a metric-specific sum loss that reflects the local structure of the respective subspace it may therefore vary greatly from one partition to another. Given that one can say, that the $M = 250$ Model reaches a plateau overall after around twelve iterations and does not improve a lot anymore.
The same can also be said about the $M = 300$ model. Although it needs much less Iterations than its counterpart, it also reaches a plateau after roughly 12 iterations and the left shifts seem to be due to the same reason as with the $M = 250$ one. This can also be checked in Figure 4.8. The fewer iterations are due to an earlier fall of the centroid shift below the given threshold.

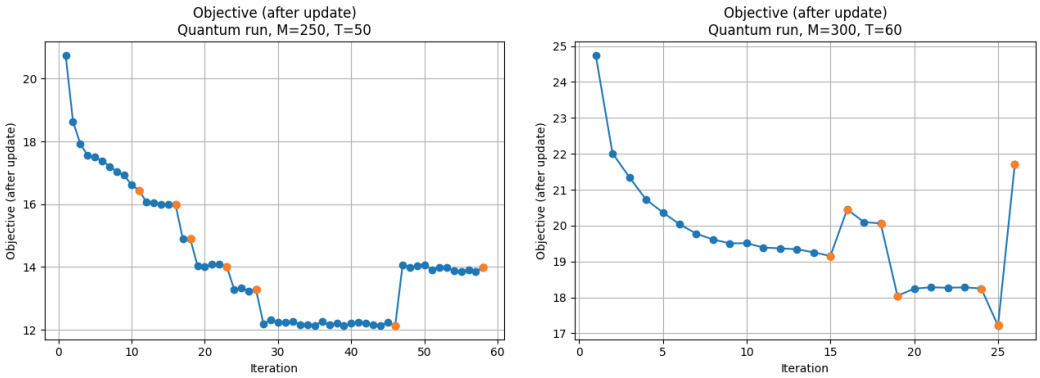


Figure 4.8 – Aggregated quantum training objective values for the M = 250 and M = 300 runs. As orange marked are the iterations which resemble the last iteration of a partition.

Across both models the diagnostic curves show the same qualitative picture. Once a plateau is reached, the accept ratio drops and more backtracking is performed. The optimiser has moved into a flat region of the log-fidelity landscape. This stems from the way the majorise–minimise procedure works. At each update the quantum k-means approximates the true loss by a linear surrogate and chooses the leading eigenvector of a weighted covariance as the candidate direction. Early on, this Rayleigh step follows the true gradient well, but near a stationary point the loss surface becomes flat in the descent direction while it remains highly curved in orthogonal directions. The surrogate then overestimates the step size or points slightly uphill. The safeguard rejects such steps and falls back to a Riemannian gradient update. Finite-shot swap-tests further add noise, so small improvements can be masked. Consequently the accept ratio collapses and the algorithm repeatedly halves the step until a slight decrease in the objective is found.

The gradient norms show a similar picture in both models as well. Corresponding to the Objective functions they go through a steep decline until they hover close above zero with only small oszillations until they reach zero. By looking at both centroid shift plots one can see quiet well how the $M = 250$ Model goes through much more trouble in optimizing the centroids than the other one. It goes to almost zero for six times, but is not stopped by the threshold as it is still higher than that. And as a short reminder, these are the averaged values. Between the first and last almost zero-shift the optimiser performs a few more steps and it looks like it might oscillate between different points. On the other hand the $M = 300$ Models centroid shift behavior has a much more linear appearance. After almost reaching zero for the first time, it performs only three more steps close to zero and then stops.

4.3 Iteration and convergence behaviour

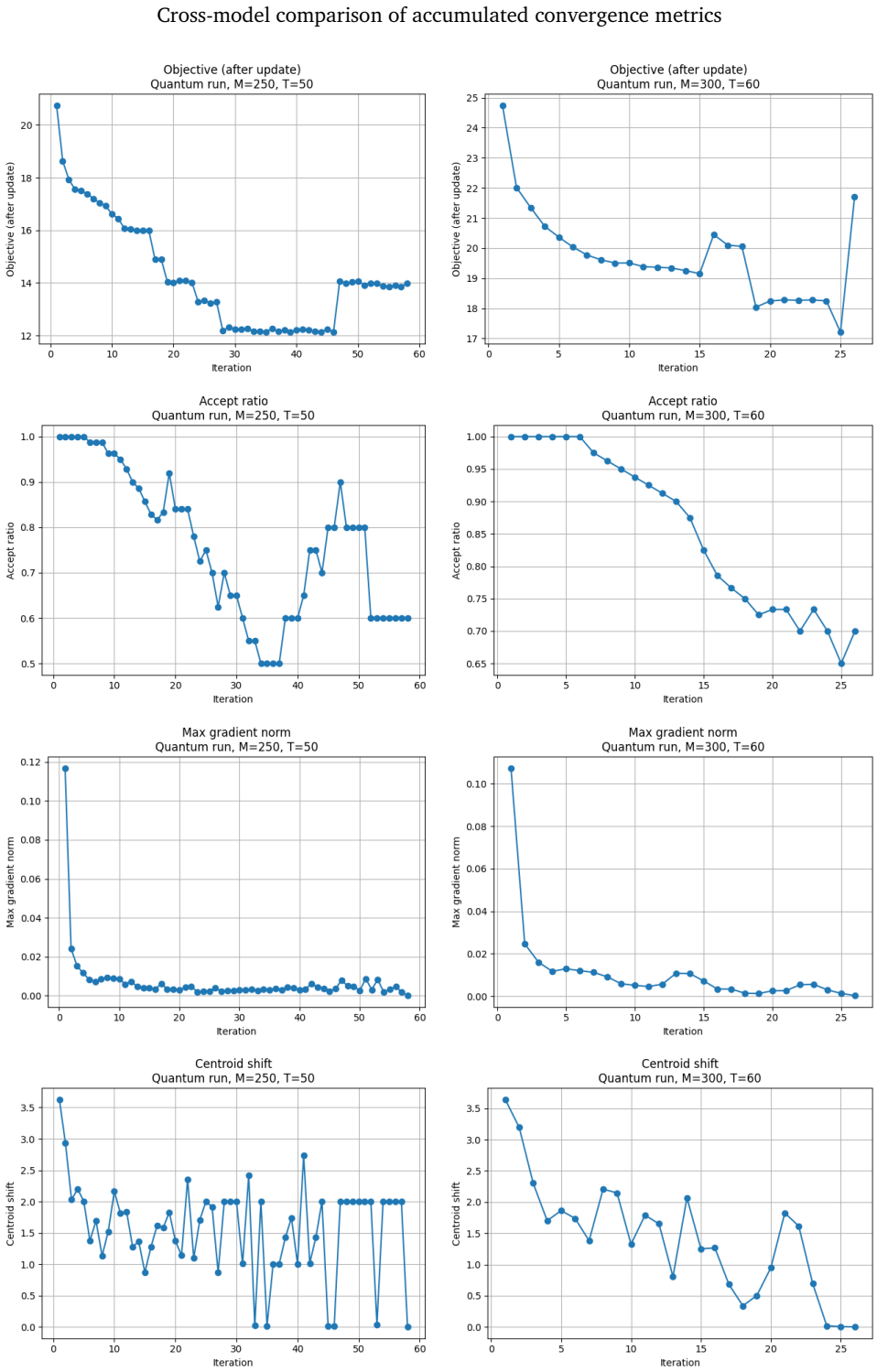


Figure 4.9 – Aggregated quantum training diagnostics for the $M = 250$ and $M = 300$ runs. Showing a) Objective value — the log-fidelity loss after each update, b) accept ratio — fraction of centroids for which the Rayleigh candidate is accepted, c) maximum gradient norm, e) centroid shift. All curves are aggregated over partitions.

In Figure 4.10 a comparison between the partition with the most iterations of either model is given. This yields a deeper insight in the models behaviour, as it does not show values which are averaged over many partitions. It shows even better that both models follow the same trend overall and also suffer from the same issue of not hitting the threshold and oscillating. Therefore it could be that the iteration anomaly of the $M = 250$ model is just an artifact of a too strict threshold, combined with an unlucky training dataset. In order to validate this hypothesis another instance of the $M = 250$ Model was trained with the same parameters as before, except, that the tolerance was set from previously $2e{-}3$ to $1e{-}2$. An overview an a discussion of the results can be found in Subsection 4.3.2.

As a last step of the detailed convergence analysis a comparison between partitions of the same model was done. A finding here is, that the training often times is dominated by one to three partitions, where the amount of iterations explode. In Figure 4.11 the convergence plots are given for partition one and partition three of the $M = 250$ model. The partition shown on the left resembles that one with the most and on the right with the fewest iterations for this model. Again one can see, that both models suffer from the same oscilattion in the centroid shift, while Partition one takes this to an extreme with eleven near zero hits. While Partition one enters a Plateau at around 10 iterations and stays in it for around 50 iterations, Partition three arguably goes into a Plateau at Iteration 9 and ending after only two more iterations. Partitions like Partition one largely dominate the models train time and for future work a finetuning of the model, so that all or most partitions behave like Partition three should be strived for.

Discussion of occasional upward steps in the inter-partition objective

A closer inspection of the per-partition history files confirms that occasional upward "jumps" of the log-fidelity objective are not violations of the theoretical monotonicity but artefacts of the measurement process. One of the largest jumps observed in the data occurs in the $M = 300$ quantum run in partition 04. the objective increases from 20.1560 to 20.4842 at iteration 20, a rise of $\Delta L \approx 0.33$. Several other partitions exhibit increases as well, but most of them stay below this large step.
In order to understand where these jumps come from, one has to take a deeper look into the characteristics of the quantum simulation and its connection to the metric. The fidelity in each sub-vector is estimated with finite shots ($N_{\text{shots}} = 7000$). The swap test yields a binomial distribution for the ancilla outcome, leading to a standard deviation of the estimated fidelity of $\sigma_F \approx 0.012$[2]. One can insert the random fluctuation directly into the loss to calculate the possible values. For a given overlap F we compute

$$L(F \pm \sigma_F) \;=\; -\ln\!\big(F \pm \sigma_F + \varepsilon\big) \quad \text{and} \quad \delta L_{\text{full}} \;=\; L(F+\sigma_F) - L(F-\sigma_F)\,.$$

This δL_{full} is the total range of possible noise-induced variation. Since the errors in F are symmetrically distributed (positive or negative deviations). For a typical *upward deviation*, half of this interval is relevant: $\delta L_{\text{half}} = \frac{1}{2}\,\delta L_{\text{full}}$. For moderate overlaps ($F = 0.20$) one finds $\delta L_{\text{half}} \approx 0.06$, for smaller fidelities ($F = 0.10$) about 0.12, and for very small overlaps ($F = 0.05$) δL_{half} increases to around 0.24. Only in the extreme case $F \approx 0.02$ does the half deviation reach about 0.65. Accordingly, the measured upward step of ≈ 0.33 still lies within the range of what is theoretically possible for very small fidelities. Most of the observed "jumps" (below 0.2) remain well below this upward step and are therefore consistent with the expected estimation noise and they do not contradict the claims in the convergence analysis.

[2]The swap test produces a binomial sample: measuring the ancilla n times yields a proportion p_0 of zeros. In the worst case $p_0 = 0.5$, the variance is $\text{Var}(p_0) = p_0(1-p_0)/n \approx 0.25/7000$, so $\sigma_{p_0} \approx 0.006$. The fidelity estimate is $\hat{F} = 2p_0 - 1$, hence $\sigma_{\hat{F}} \approx 2\sigma_{p_0} \approx 0.012$.

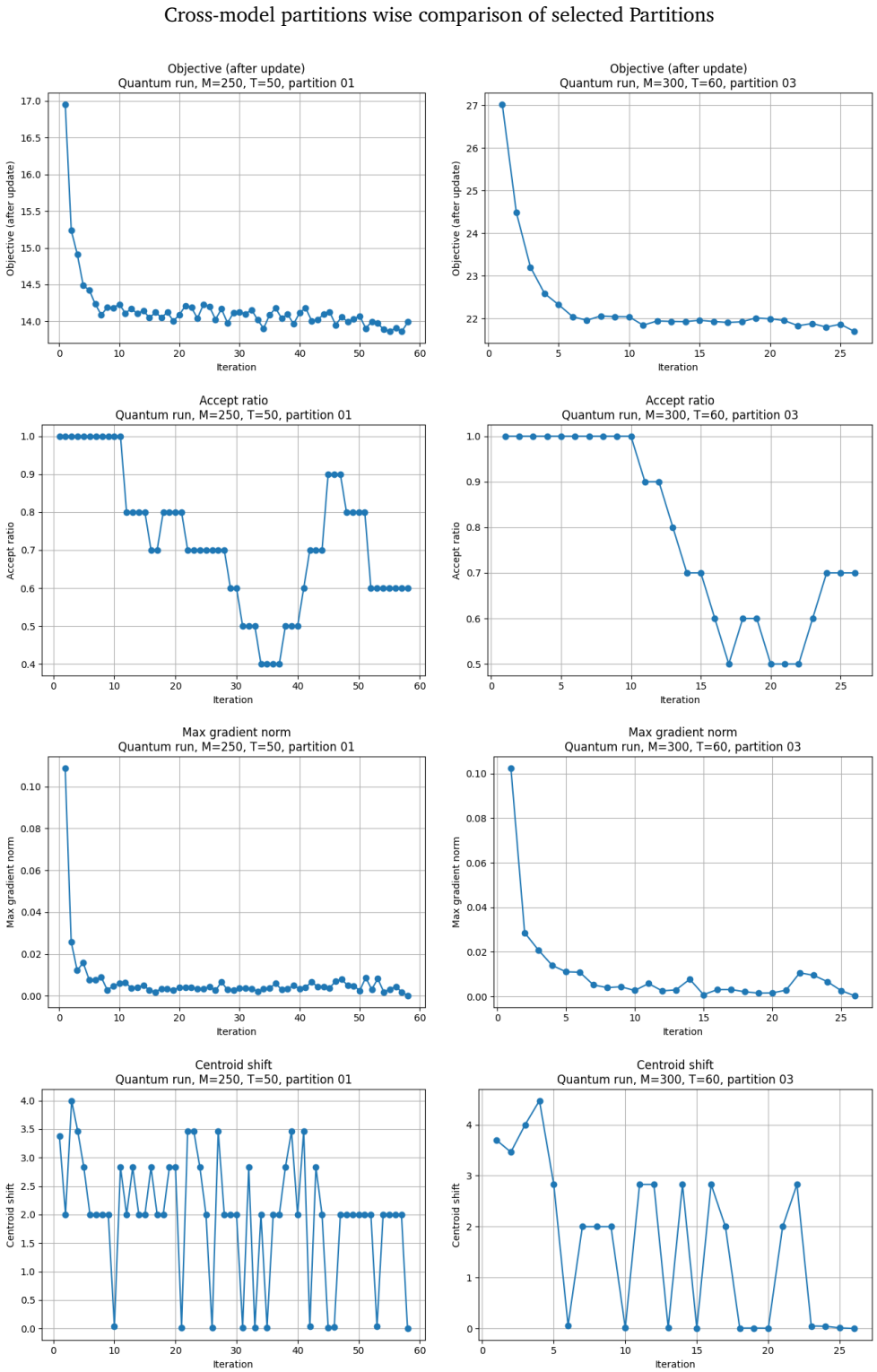


Figure 4.10 – Training diagnostics for the $M = 250$ - Partition 1 and $M = 300$ - Partition 3 runs. Showing a) Objective value — the log-fidelity loss after each update, b) accept ratio — fraction of centroids for which the Rayleigh candidate is accepted, c) maximum gradient norm, d) centroid shift.

Cross-Partition comparison for the $M = 250$ Models best and worst performing Partition

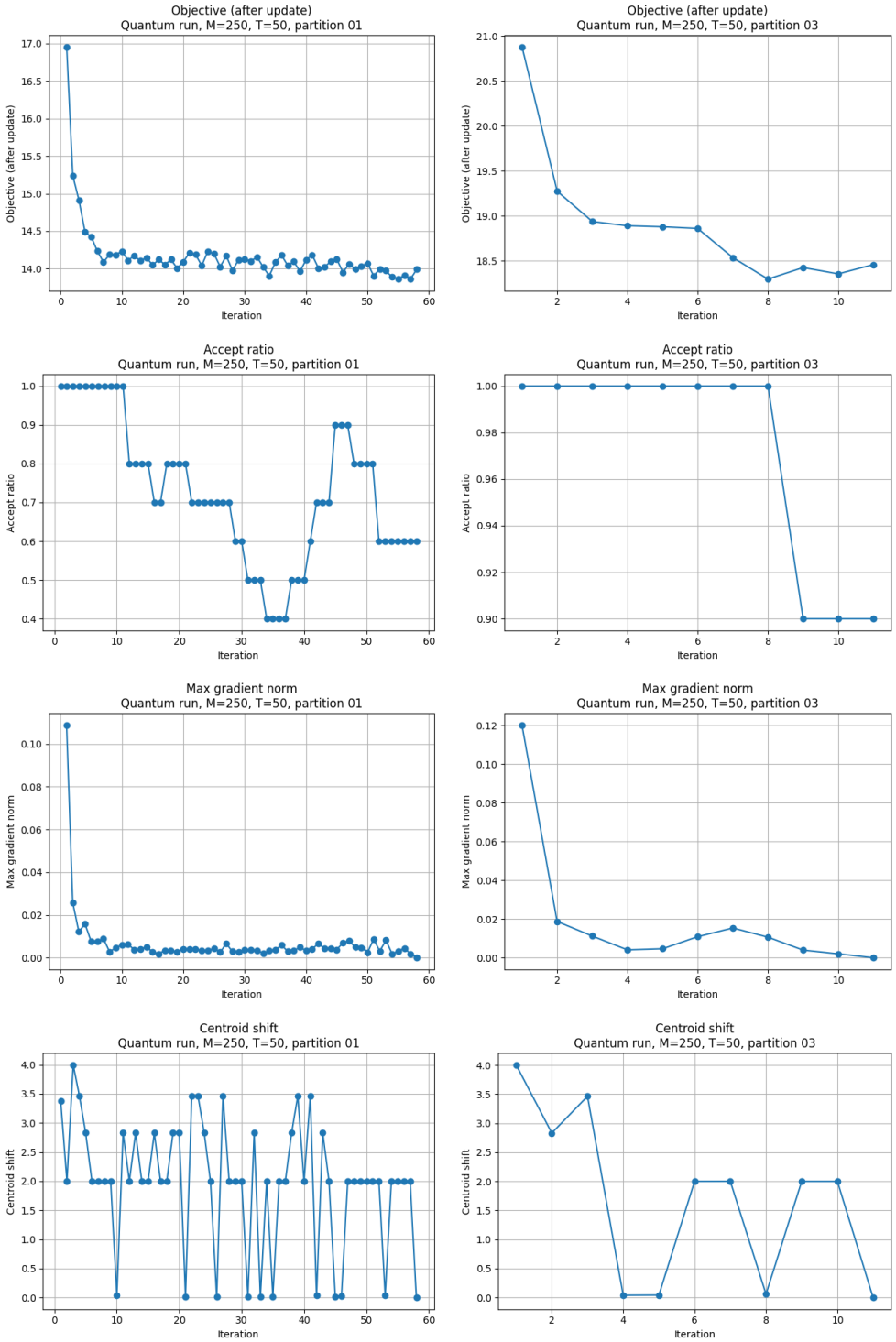


Figure 4.11 – Training diagnostics for the $M = 250$ - Partition 1 and 3 runs. Showing a) Objective value — the log-fidelity loss after each update, b) accept ratio — fraction of centroids for which the Rayleigh candidate is accepted, c) maximum gradient norm, d) centroid shift.

4.3.2 Testing with higher tolerance

As derived from the previous subsection the trained models might suffer from a too strict tolerance setting, resulting in iteration count anomalys like with the $M = 250$ model. In order to test the influence of the tolerance setting on the models behaviour another instance of the $M = 250$ model was trained with a tolerance setting of 10^{-2} instead of $2 \cdot 10^{-3}$.

The results of this are surprisingly good. In comparison to the previous model the mean number of iterations per partition fell from 29 to 15 and by that perfectly aligns with the general trend of the models mean iterations previously shown shown in Figure 4.6. Meanwhile the accuracy even improved from previously 92% to 94%. As a reminder, the classical model at $M = 250$ also had an accuracy of 94%. So both models accuracy already converge consistently at the $M = 250$ under these parameters. In Figure 4.12, 4.13 a comparison of the already introduced training metrics between Partition 1 of both models is given. Partition 1 was chosen, because that is the Partition with the most iterations in the old version. Here the achieved improvements are well visible. One might ask, how it comes, that there are differences in the plots even within the first few iterations, though in theory, adjusting the convergence threshold should only affect the stopping time of the algorithm. In practice, however, this can again be explained with the stochastic swap-tests. If one runs the same model twice with different tolerances, the sequence of backtracking steps and swap-test calls can change, as well as the intitial centroid intialisation. With a tighter tolerance the algorithm performs more Rayleigh and gradient steps before halting, and thus consumes random numbers in a different order. These differences propagate immediately – two runs will generate slightly different fidelity estimates and therefore different centroid shifts and other metrics, even in the very first iterations. Consequently, the initial points of the plotted metric curves are not identical across runs with different tolerances.

The overall earlier convergence, leading to fewer needed iterations with a higher acceptance rate and an improvement in accuracy, instead of a deterioration as one might intuitively think, speak in favor for a further investigation into the adjustment of this hyperparameter. Due to time reasons this was omitted from this thesis, but for future research a reasonable approach would be to first search for ideal tolerance settings and then perform larger numerical tests on the algorithm.

Comparing Partition 1 of the old an new $M = 250$ Model

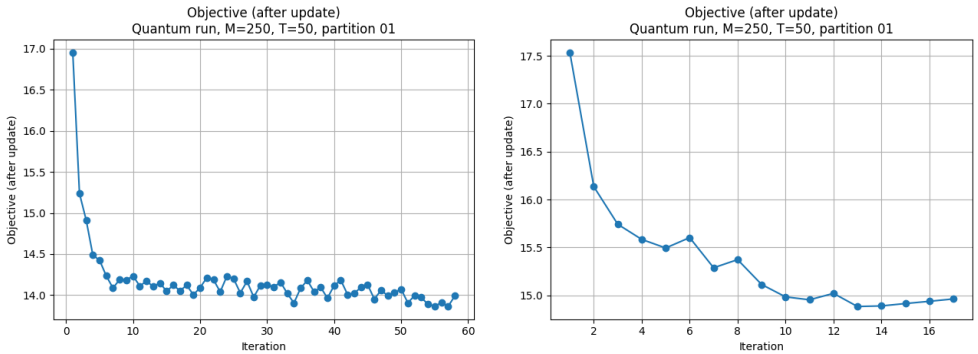


Figure 4.12 – Training diagnostics for the $M = 250$ - Partition 1 Model with old tolerance settings on the left and new settings on the right. Showing the Objective value — the log-fidelity loss after each update

Comparing Partition 1 of the old an new $M = 250$ Model

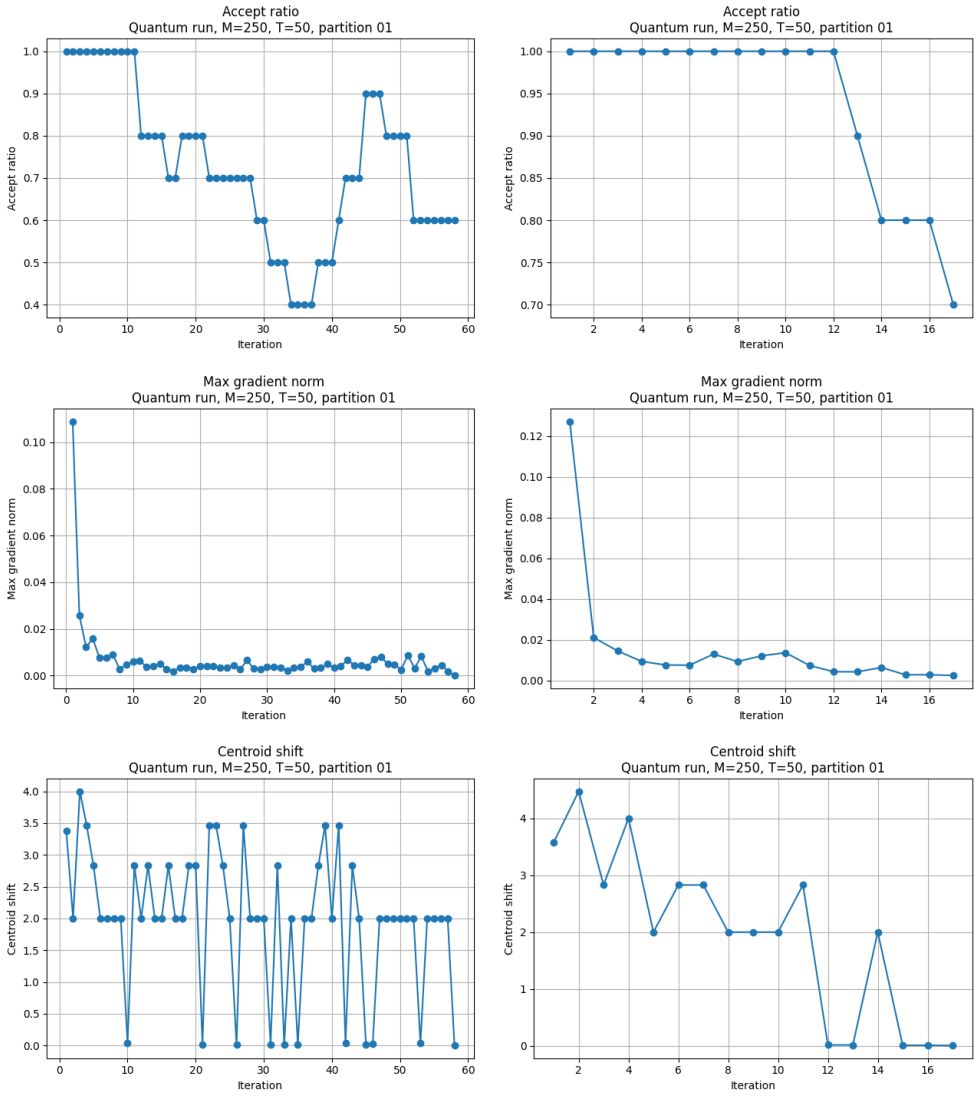


Figure 4.13 – Training diagnostics for the $M = 250$ - Partition 1 Model with old tolerance settings on the left and new settings on the right. Showing a) accept ratio — fraction of centroids for which the Rayleigh candidate is accepted, b) maximum gradient norm, c) centroid shift.

4.4 Cluster quality: inertia and log-fidelity loss

As a last analysis step the mean cluster losses for classical and quantum runs are compared in Figure 4.14. The classical loss (inertia) grows monotonically with M because more points contribute to the sum of squared errors. When normalised per point the inertia remains approximately constant at ≈ 0.07, indicating that the average distance of a sample to its assigned centroid is stable across data sizes. The quantum loss, measured as the sum of negative log-fidelities, also increases with M,

but its per-point normalised value hovers around 0.49–0.52 (viewing the first point as an outlier). Eventhough there is more variabillity in the quantum mean cluster loss, it shows that the average overlap between data and centroid vectors remains roughly constant as more samples are added. Though the two metrics are not directly comparable, both suggest that cluster quality scales well with data volume, as adding more training points does not significantly degrade the per-sample error in either metric.

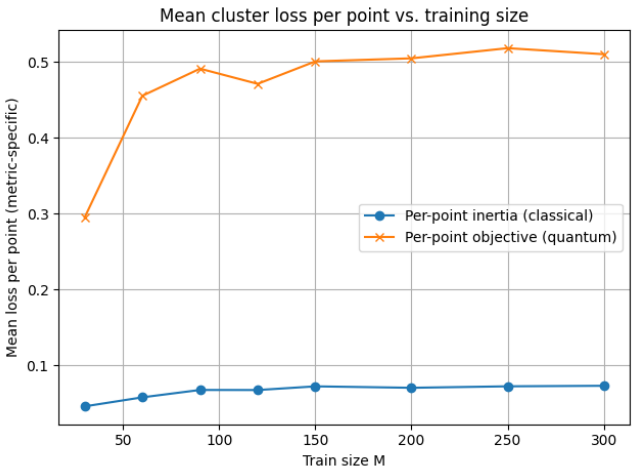


Figure 4.14 – Mean cluster loss per training point vs. M. For classical runs this is the inertia per point. For quantum runs it is the log-fidelity objective per point. Both curves are nearly flat, indicating that cluster quality does not deteriorate with increased training data.

Discussion and Conclusions

The experiments presented in this chapter provide an initial empirical comparison between a classical and a hybrid–quantum implementation of the product–quantisation k-nearest–neighbour (PQ-kNN) algorithm. The main findings can be summarised as follows.

Classification performance.

The classical pipeline already achieves high accuracy with moderate training sizes M. Its accuracy rises from 0.33 at $M = 30$ to 0.95 at $M = 300$. Under the old tolerance setting the hybrid quantum pipeline exhibits a similar monotonic improvement, but starts more slowly and only matches the classical accuracy for $M = 300$. These results indicate that the additional complexity of the quantum pipeline does not translate into immediate accuracy gains on small datasets, but that comparable classification performance can be reached when sufficiently many training samples are available. The lower accuracy can also be explained with the statistical nature of the swap test, possibly leading to worse clustering results and convergence behaviour.

Convergence behaviour.

The average number of training iterations per partition increases with M for both algorithms, but remains much lower for the classical variant. For example, the classical algorithm needs 2.5 iterations

on average at $M = 30$ and 13.8 iterations at $M = 300$, whereas the hybrid quantum variant requires 4.9 and 20.1 iterations, respectively. Detailed diagnostics reveal that the quantum optimiser often experiences a sharp decrease in the log-fidelity objective during the first ten to twenty iterations, followed by plateaus where improvements are marginal or the model even worsens. In this region the surrogate model used by the majorise–minimise (MM) scheme runs into a troubling area, leading to rejected candidates and frequent backtracking. As a consequence, the accept ratio drops and the algorithm relies on the safeguarded Riemannian gradient step, which additionally slows down convergence.

Partition-specific behaviour.

In each model often a few partitions exhibit a large number of iterations, while the majority converge quicker. This heterogeneity explains the larger mean iteration count and suggests that an overly strict global tolerance can cause single partitions to dominate the overall training time.

Measurement noise.

Occasional increases in the log-fidelity objective do not indicate algorithmic failure but can be caused by statistical fluctuations in the finite-shot SWAP tests. With 7 000 shots per fidelity estimate, the standard deviation of the estimated fidelity is approximately 0.012. Inserting this fluctuation into the loss calculation is sufficient to explain apparent spikes.

Conclusions and outlook.

It was possible to demonstrate that the hybrid quantum version can, in principle, reach similar classification accuracy as its classical counterpart, but currently does so with more complex convergence behaviour. Key obstacles to practical applicability are the long training times on classical machines, which will be used until much more sophisticated quantum machines are available and the sensitivity of the majorise–minimise optimisation to flat regions in the log-fidelity landscape. To address these challenges, future work could explore the following directions:

- *Adaptive or ideal tolerances.* Allowing partition-specific stopping criteria or dynamically adjusting the centroid-shift threshold may prevent a small number of problematic partitions from dominating the training process. Another way would be to try to find an ideal solution by taking a look at the possible variance caused by the statistical SWAP test.
- *Hyperparameter sweeps.* Explore (n, c, k) variations and inspect their influence on the algorithms behaviour.
- *Parallelisation.* Because the product-quantisation partitions are independent, distributing them across multiple cores or processing units could reduce total training.
- *Alternative optimisation schemes.* Replacing the MM approach with optimisation algorithms that adaptively scale the step size or combine gradient and surrogate information could improve convergence in flat regions of the objective.
- *Robust estimators.* Variance-reduction techniques, such as shot-recycling or increasing the number of shots only in later iterations, may mitigate the impact of measurement noise.
- *Extended experiments.* Evaluating the algorithms on larger and more diverse datasets, increasing the number of training sizes M, and comparing against other quantum and classical clustering methods will provide a more comprehensive assessment of the PQ-kNN approach.

Overall, the present chapter should be viewed as a proof of concept rather than a definitive benchmark. While the hybrid quantum PQ-kNN algorithm does not yet offer a practical advantage over the classical baseline, the insights gained here—particularly regarding convergence diagnostics and the influence of partitions—highlight possible avenues for algorithmic and implementation-level improvements.

5 CONCLUSION

This thesis introduced a hybrid quantum–classical Product Quantization k-NN pipeline grounded in the log-fidelity k-Means procedure. The key design choices — amplitude encoding for sub-vectors, fidelity estimation via the Swap Test, log-fidelity as a loss function and a centroid update that couples a Rayleigh–Ritz (eigenvector) surrogate with the safeguarded Riemannian gradient step — are tailored to Product Quantization (PQ). Crucially, taking the negative logarithm of fidelity renders the global objective *additive* across PQ partitions, aligning the quantum distance measure with the structure of PQ.

On the *Digits* Dataset with fixed parameters (64-D, $n=8$ partitions, $c=10$ sub-centroids, $k=9$), the classical PQ baseline achieves strong accuracy with moderate training sizes. The hybrid quantum variant tracks the same trend, although it is typically a few percentage points lower on smaller training sets, but closes the gap as data increase. For $M=300$ both reach 0.95 accuracy. Confusion matrices at $M=300$ are identical, indicating that the quantum distance does not change decision boundaries at larger scale. A tolerance ablation at $M=250$ (relaxing the centroid-shift threshold) halved the mean number of quantum k-Means iterations and improved accuracy to the classical level, suggesting that improved convergence sensitivity settings can be beneficial for the algorithms performance. Occasional upward steps of the objective can be explained with finite-shot estimation noise rather than algorithmic failures. Across runs, per-point cluster losses remain approximately stable as M grows, indicating that cluster quality scales well. Under current hardware a main disadvantage remains to be the long runtime and resource expensive implementation with the given quantum simulations, as there is no quantum hardware available which could even remotely implement this algorithm. Another caveat is the small amount of trained and tested models. Further, numerically larger tests need to be conducted!

Future work

- *Tuning and testing:* Systematic tests of tolerance and parameter settings. Training and testing of more models on larger and more diverse datasets.
- *Parallelism:* Implementing parallel training across partitions.
- *Quantum subroutines:* Grover minimum finding for assignment. QRAM-based state preparation when available. Exploratory quantum spectral routines for the centroid eigenproblem.
- *Heuristics/improved centroid update:* Search for a computationally easier centroid update routine or use the given one and try to derive a useful heuristic from it, with less mathematical guarantees.

In sum, this thesis delivers the first mathematically coherent PQ design, that lives within Quantuminformation. The in Chapter 2.2, 3 and Appendix 6.1 theoretically derived design choices proofed to perform comparably to the classical variant. Major subroutines of it could in Theory be implemented on a quantum machine and the additive distance, convergence-minded centroid update, and empirical study hand a starting point from where future work on this Problem can be conducted.

APPENDIX 6

6.1 Outline for a convergence proof

This section presents a version of the previously stated convergence analysis. It incorporates both the algorithmic details of the Python implementation and closes some mathematical gaps.

Let normalized data points $\mathscr{D} = \{\psi_i\}_{i=1}^N \subset \mathbb{C}^d$ and $k \in \mathbb{N}$ be given, where $N := |\mathscr{D}|$ denotes the number of data points. The goal is to partition the objects ψ_i into k clusters and to determine a center $\phi_j \in \mathbb{S}^{2d-1} = \{c \in \mathbb{C}^d : \|c\| = 1\}$ for each cluster. For a center ϕ we define a smoothed logarithmic distance

$$d_{\log}^{\varepsilon}(\psi, \phi) := -\ln\big(|\langle\psi, \phi\rangle|^2 + \varepsilon\big),$$

where $\varepsilon > 0$ is a fixed regularisation constant preventing division by zero and numerical instabilities. The overall cost function is then

$$L(\boldsymbol{\ell}, \boldsymbol{\phi}) = \sum_{i=1}^{N} d_{\ln}^{\varepsilon}\big(\psi_i, \phi_{\ell_i}\big),$$

where $\boldsymbol{\ell} = (\ell_1, \ldots, \ell_N) \in \{1, \ldots, k\}^N$ are the cluster assignments and $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_k) \in (\mathbb{S}^{2d-1})^k$ the centers.

6.1.1 Algorithm and implementation details

The algorithm extends classical Lloyd iteration by a re-weighted eigenvector step (Rayleigh–Ritz) and a safeguarded Riemannian gradient descent. The following steps are iterated (see also the implementation in the class `QuantumKMeans`[3]):

1. *Initialization.* The initial centers are selected via k-means++. All subsequent computations use a fixed regularisation parameter $\varepsilon > 0$, and the weights in the update step are scaled so that they sum to one; this improves numerical stability and prevents individual weights from blowing up.

2. *Assignment step.* Given centers $\boldsymbol{\phi}^{(t)}$, each data point is assigned to the nearest center:

$$\ell_i^{(t)} \in \arg\min_{1 \le j \le k} d_{\ln}^{\varepsilon}\big(\psi_i, \phi_j^{(t)}\big).$$

[3]The class `QuantumKMeans` implements the algorithm described here. In particular, the function `_centroid_update` performs the center update.

Since the centers are fixed, this choice minimizes the cost function over all assignments and cannot increase the loss. In the implementation, empty clusters are not reinitialized randomly. Instead, their centers remain unchanged. This ensures that the loss stays constant in this case, preserving monotonic decrease.

3. *Center update.* For each $j \in \{1,\dots,k\}$ let

$$C_j^{(t)} := \left\{ i \in \{1,\dots,N\} : \ \ell_i^{(t)} = j \right\}$$

be the index set of cluster j. If $C_j^{(t)} = \varnothing$, then $\phi_j^{(t+1)} = \phi_j^{(t)}$. Otherwise:

a) Minorant construction.

Let $c_0 = \phi_j^{(t)}$. For each point $i \in C_j^{(t)}$ define an unnormalised weight

$$\hat{w}_i(c_0) := \frac{1}{|\langle \psi_i, c_0\rangle|^2 + \varepsilon}, \quad w_i(c_0) := \frac{\hat{w}_i(c_0)}{\sum_{m \in C_j^{(t)}} \hat{w}_m(c_0)}.$$

By normalisation $\sum_{i \in C_j^{(t)}} w_i(c_0) = 1$. Define the self-adjoint operator (a weight-weighted covariance matrix)

$$S_j(c_0) \ := \ \sum_{i \in C_j^{(t)}} w_i(c_0)\, \psi_i \psi_i^\dagger.$$

Define the cluster loss

$$f_j(c) \ := \ -\sum_{i \in C_j^{(t)}} \ln\!\big(|\langle \psi_i, c\rangle|^2 + \varepsilon\big).$$

Because $x \mapsto -\ln(x+\varepsilon)$ is strictly convex, we have the first-order underestimate

$$-\ln(x+\varepsilon) \geq -\ln(x_0+\varepsilon) - \frac{x - x_0}{x_0 + \varepsilon}$$

for $x, x_0 > 0$. Setting $x = |\langle \psi_i, c\rangle|^2$ and $x_0 = |\langle \psi_i, c_0\rangle|^2$ defines the minorant

$$h_j(c; c_0) = -\sum_{i \in C_j^{(t)}} \Big[\ln\!\big(|\langle \psi_i, c_0\rangle|^2 + \varepsilon\big) + \hat{w}_i(c_0)\big(|\langle \psi_i, c\rangle|^2 - |\langle \psi_i, c_0\rangle|^2\big) \Big].$$

By collecting constants one can write $h_j(c; c_0) = \text{const} - c^\dagger S_j(c_0) c$. Hence $f_j(c) \geq h_j(c; c_0)$ and $f_j(c_0) = h_j(c_0; c_0)$.

b) Rayleigh–Ritz step.

Since $S_j(c_0)$ is self-adjoint and positive semidefinite, minimizing $h_j(c; c_0)$ subject to $\|c\| = 1$ is equivalent to maximizing the Rayleigh quotient $c^\dagger S_j(c_0) c$. For a Hermitian matrix A the Rayleigh quotient $\langle v, Av\rangle$ achieves its maximum precisely when v is an eigenvector corresponding to the largest eigenvalue[Roc20]. Let $u \in \mathbb{S}^{2d-1}$ be a normalised eigenvector of $S_j(c_0)$ for the largest eigenvalue. Then

$$h_j(u; c_0) \ = \ \min_{\|c\|=1} h_j(c; c_0).$$

This vector u is chosen as a surrogate candidate.

c) Accept/reject test.

The true loss $f_j(u)$ is compared with $f_j(c_0)$. If $f_j(u) \le f_j(c_0)$, then $\phi_j^{(t+1)} := u$. Otherwise u is rejected, and a Riemannian gradient descent is performed.

d) Riemannian gradient descent with adaptive step size.

Let $c_0 = \phi_j^{(t)}$. The Euclidean gradient of f_j at c_0 is

$$\nabla f_j(c_0) = -2 \sum_{i \in C_j^{(t)}} \frac{\mathrm{Re}\big(\langle \psi_i, c_0 \rangle \, \psi_i\big)}{|\langle \psi_i, c_0 \rangle|^2 + \varepsilon}.$$

It is projected orthogonally onto the tangent space $T_{c_0}\mathbb{S}^{2d-1} = \{v : \langle v, c_0 \rangle = 0\}$:

$$\operatorname{grad} f_j(c_0) = \nabla f_j(c_0) - \langle \nabla f_j(c_0), c_0 \rangle \, c_0.$$

It is known that the negative Riemannian gradient provides the direction of steepest descent and that sufficiently small steps decrease the function[AMS09]. Consider the curve

$$\gamma(\eta) = \frac{c_0 - \eta \operatorname{grad} f_j(c_0)}{\| c_0 - \eta \operatorname{grad} f_j(c_0) \|}.$$

A Taylor expansion yields

$$\frac{\mathrm{d}}{\mathrm{d}\eta} f_j\big(\gamma(\eta)\big)\Big|_{\eta=0} = -\| \operatorname{grad} f_j(c_0) \|^2 < 0,$$

implying that $f_j(\gamma(\eta)) < f_j(c_0)$ for sufficiently small $\eta > 0$. The implementation performs an adaptive backtracking line search: starting with $\eta = 1$, the step size is halved until $f_j(c_0 - \eta \operatorname{grad} f_j(c_0)) < f_j(c_0)$. As the Riemannian gradient is Lipschitz-continuous on the compact sphere[RS24], a sufficiently small step always exists.

6.1.2 Monotonic decrease of the cost

Proposition 1 (Monotonic decrease)**.** *Let $(\boldsymbol{\ell}^{(t)}, \boldsymbol{\phi}^{(t)})_{t \ge 0}$ be the sequence generated by the above algorithm. Then for all $t \ge 0$*

$$L\big(\boldsymbol{\ell}^{(t+1)}, \boldsymbol{\phi}^{(t+1)}\big) \;\le\; L\big(\boldsymbol{\ell}^{(t)}, \boldsymbol{\phi}^{(t)}\big),$$

with strict inequality whenever at least one cluster is non-empty and in step (2) the surrogate candidate is accepted or a gradient step is executed. Empty clusters retain their center, so the loss remains constant.

Proof. Assignment step. Fixing the centers, the assignment $\boldsymbol{\ell}^{(t)}$ minimizes $L(\boldsymbol{\ell}, \boldsymbol{\phi}^{(t)})$ over all assignments. Hence

$$L\big(\boldsymbol{\ell}^{(t)}, \boldsymbol{\phi}^{(t)}\big) = \min_{\boldsymbol{\ell}} L\big(\boldsymbol{\ell}, \boldsymbol{\phi}^{(t)}\big) \;\le\; L\big(\boldsymbol{\ell}^{(t-1)}, \boldsymbol{\phi}^{(t)}\big).$$

In the implementation, empty clusters retain their old centers, so the loss does not increase.

center update. Let j be a non-empty cluster. As shown above, $h_j(c; c_0)$ is a linear underestimator of f_j with $f_j(c_0) = h_j(c_0; c_0)$. The normalized eigenvector u of $S_j(c_0)$ corresponding to the largest eigenvalue minimizes h_j[Roc20]. If $f_j(u) \le f_j(c_0)$, then $\phi_j^{(t+1)} = u$ and the loss does not increase. Otherwise a Riemannian gradient step with backtracking is performed. As discussed above, there exists a step size $\eta > 0$ such that $f_j(c_0 - \eta \operatorname{grad} f_j(c_0)) < f_j(c_0)$[RS24]. The implementation halves

the step size until this condition holds. Hence the center of cluster j is either left unchanged or updated so that f_j strictly decreases. The losses of the other clusters are unchanged. Therefore the total cost does not increase and decreases strictly whenever at least one true update occurs.

Since $|\langle\psi_i, \phi_{\ell_i}\rangle|^2 \in [0,1]$, we have $-\ln\big(|\langle\psi_i, \phi_{\ell_i}\rangle|^2 + \varepsilon\big) \geq -\ln(1+\varepsilon)$ for each i. Summing over all data points yields

$$L\big(\boldsymbol{\ell}^{(t)}, \boldsymbol{\phi}^{(t)}\big) = \sum_{i=1}^{N} \big(-\ln\big(|\langle\psi_i, \phi_{\ell_i}\rangle|^2 + \varepsilon\big)\big) \geq -N \ln(1+\varepsilon).$$

The non-increasing sequence $(L^{(t)})_{t\geq 0}$ is thus bounded below and converges.

□

6.1.3 Quadratic surrogate with weights

Lemma 1 (Weighted quadratic surrogate). *Let $c_0 \in \mathbb{S}^{2d-1}$ and let $w_i(c_0)$ and $S_j(c_0)$ be defined as above. Then*

$$h_j(c; c_0) = -\sum_{i \in C_j^{(t)}} \Big[\ln\big(|\langle\psi_i, c_0\rangle|^2 + \varepsilon\big) + \hat{w}_i(c_0)\big(|\langle\psi_i, c\rangle|^2 - |\langle\psi_i, c_0\rangle|^2\big)\Big]$$

satisfies $h_j(c; c_0) \leq f_j(c)$ for all $c \in \mathbb{S}^{2d-1}$ and $h_j(c_0; c_0) = f_j(c_0)$. Subject to $\|c\| = 1$, $h_j(c; c_0)$ is uniquely minimized by the normalized eigenvector of $S_j(c_0)$ corresponding to the largest eigenvalue.

Proof. The linear underestimation of $-\ln(x+\varepsilon)$ shows $f_j(c) \geq h_j(c; c_0)$ and $f_j(c_0) = h_j(c_0; c_0)$. As shown above, $h_j(c; c_0) = \text{const} - c^\dagger S_j(c_0) c$. Minimizing over $\|c\| = 1$ reduces to maximizing the Rayleigh quotient $c^\dagger S_j(c_0) c$, which is achieved by the eigenvector associated with the largest eigenvalue[Roc20]. □

6.1.4 Local descent on the sphere

Lemma 2 (Local Riemannian gradient descent). *Let $f : \mathbb{S}^{2d-1} \to \mathbb{R}$ be continuously differentiable, $c \in \mathbb{S}^{2d-1}$ and $\operatorname{grad} f(c) \neq 0$ the Riemannian gradient, i.e. the orthogonal projection of the Euclidean gradient onto the tangent space $T_c\mathbb{S}^{2d-1}$. For the curve*

$$\gamma(\eta) = \frac{c - \eta \operatorname{grad} f(c)}{\big\|c - \eta \operatorname{grad} f(c)\big\|}$$

there exists $\eta_0 > 0$ such that

$$\frac{\mathrm{d}}{\mathrm{d}\eta} f\big(\gamma(\eta)\big)\Big|_{\eta=0} = -\|\operatorname{grad} f(c)\|^2 < 0,$$

and hence $f(\gamma(\eta)) < f(c)$ for all $0 < \eta < \eta_0$. This shows that the negative Riemannian gradient is the direction of steepest descent[AMS09].

Proof. The curve $\gamma(\eta)$ lies on the sphere for sufficiently small η and satisfies $\gamma(0) = c$. By the chain rule,

$$\frac{\mathrm{d}}{\mathrm{d}\eta} f\big(\gamma(\eta)\big)\Big|_{\eta=0} = \langle \nabla f(c), \dot{\gamma}(0)\rangle,$$

where $\dot{\gamma}(0) = -\operatorname{grad} f(c)$. Since $\langle \nabla f(c), \operatorname{grad} f(c)\rangle = \|\operatorname{grad} f(c)\|^2$, the derivative at zero is negative. A first-order Taylor expansion then shows that $f(\gamma(\eta)) < f(c)$ for sufficiently small $\eta > 0$. On the compact sphere the Riemannian gradient is Lipschitz-continuous, so an admissible step size always exists[RS24]. □

6.1.5 Stationarity of accumulation points

Theorem 1 (Stationarity). *The sequence of centers $\{\boldsymbol{\phi}^{(t)}\}_{t\geq 0}$ has at least one accumulation point $\boldsymbol{\phi}^* \in (\mathbb{S}^{2d-1})^k$. For every non-empty cluster j we have* $\operatorname{grad} f_j(\phi_j^*) = 0$*, i.e. these accumulation points are blockwise stationary.*

Proof. The product manifold $(\mathbb{S}^{2d-1})^k$ is compact, so $\{\boldsymbol{\phi}^{(t)}\}$ has a convergent subsequence. There are only finitely many cluster assignments, so one can choose a subsequence along which $\boldsymbol{\ell}^{(t)}$ is constant. On this subsequence the centers are updated only by step (2). By the monotonic decrease proposition, the loss sequence converges. There cannot be infinitely many strict decreases, otherwise L would diverge to $-\infty$. Suppose there were a non-empty cluster j with $\operatorname{grad} f_j(\phi_j^*) \neq 0$. Then the previous lemma would provide a neighborhood in which a Riemannian gradient step strictly reduces the loss. Since the implementation's backtracking line search finds such a reduction, this would contradict the convergence of the loss sequence. Hence $\operatorname{grad} f_j(\phi_j^*) = 0$. □

6.1.6 Remarks on practical implementation and robustness

- *Handling of empty clusters.* In the implementation, empty clusters retain their old center. Alternatively, one could relocate an empty center into the largest cluster to avoid empty clusters, provided this does not increase the loss.
- *Weights and numerical stability.* The weights $w_i(c_0)$ are normalized so that their sum is one. Without normalisation, individual weights could become very large, causing numerical instability. Weights may also be heuristically clipped to avoid over-emphasising single points.
- *Step size rule.* Instead of a fixed small step size, a backtracking line search is used. The step size is halved until a decrease is observed. This adaptive choice is robust against variations in the Lipschitz constant of the gradient.
- *Initialisation and multiple runs.* Due to non-convexity, the algorithm may converge to local minima. It is advisable to run multiple k-means++ initialisations with different random seeds and select the best solution.
- *Accept/reject threshold.* In practice one may introduce a threshold $\tau > 0$ and accept a surrogate candidate if $f_j(u) \leq f_j(c_0) - \tau$. This can dampen numerical fluctuations.

6.2 Code of the centroid update

```
def _centroid_update(self, X: np.ndarray, labels: np.ndarray):
    eps_num = 1e-12
    newC = np.zeros_like(self.cluster_centers_)

    accept_count = 0
    backtracks: List[int] = []
    grad_norms: List[float] = []
    cluster_sizes: List[int] = []
    cluster_losses: List[float] = []

    for k in trange(self.n_clusters, desc="Centroid-update", ↘
        leave=False):
```

```
12          pts = X[labels == k]
13          cluster_sizes.append(int(len(pts)))
14          if len(pts) == 0:
15              newC[k] = self.cluster_centers_[k]
16              cluster_losses.append(0.0)
17              backtracks.append(0)
18              grad_norms.append(0.0)
19              continue
20
21          if self.distance_metric == "log_fidelity":
22              c_old = self.cluster_centers_[k]
23
24              # Re-weighted EV-candidate
25              overlaps = np.abs(pts @ c_old)
26              F = overlaps ** 2
27              w = 1.0 / (F + self.smooth_eps)
28              w /= np.sum(w)
29              Sigma = (pts.T * w) @ pts     # PSD, hermitisch
30
31              # Riemann-Gradienten-Norm am alten Punkt (bis auf ↘
                    Konstante)
32              Sc = Sigma @ c_old
33              # Projektion: (I - c c^T) Sc
34              proj = Sc - (np.vdot(c_old, Sc).real) * c_old
35              grad_norms.append(float(2.0 * np.linalg.norm(proj)))
36
37              # Leit-EV
38              eigvals, eigvecs = np.linalg.eigh(Sigma)
39              m = eigvecs[:, np.argmax(eigvals)]
40              c_cand = m / (np.linalg.norm(m) + eps_num)
41
42              # Echte Cluster-Losses
43              f_old = self._cluster_log_fid_loss(pts, c_old, ↘
                    self.smooth_eps)
44              f_cand = self._cluster_log_fid_loss(pts, c_cand, ↘
                    self.smooth_eps)
45
46              if f_cand <= f_old:
47                  newC[k] = c_cand
48                  accept_count += 1
49                  backtracks.append(0)
50              else:
51                  # Fallback: Backtracking alongside negative ↘
                        riemannian gradient
52                  d = proj
53                  d_norm = np.linalg.norm(d)
54                  if d_norm < 1e-12:
55                      newC[k] = c_old
56                      backtracks.append(0)
57                  else:
58                      d = d / d_norm
59                      step = 1.0
```

```
60                      bt = 0
61                      accepted = False
62                      for _ in range(10):  # 10 halvings should ↘
                            be enough
63                          bt += 1
64                          c_try = c_old + step * d
65                          c_try /= (np.linalg.norm(c_try) + ↘
                                eps_num)
66                          f_try = ↘
                                self._cluster_log_fid_loss(pts, ↘
                                c_try, self.smooth_eps)
67                          if f_try < f_old:
68                              newC[k] = c_try
69                              accepted = True
70                              break
71                          step *= 0.5
72                      if accepted:
73                          backtracks.append(bt)
74                      else:
75                          newC[k] = c_old
76                          backtracks.append(bt)
77
78              # Loss of new reporting
79              cluster_losses.append(self._cluster_log_fid_loss(pts, ↘
                    newC[k], self.smooth_eps))
80
81          else:
82              # one_minus_fidelity: classical mean + normalization
83              m = np.mean(pts, axis=0)
84              c_new = m / (np.linalg.norm(m) + eps_num)
85              newC[k] = c_new
86              cluster_losses.append(0.0)
87              backtracks.append(0)
88              grad_norms.append(0.0)
89
90      stats = {
91          "accept_count": accept_count,
92          "backtracks": backtracks,
93          "grad_norms": grad_norms,
94          "cluster_sizes": cluster_sizes,
95          "cluster_losses": cluster_losses,
96      }
97      return newC, stats
```

Listing 6.1 – Safeguarded centroid update in `QuantumKMeans`

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

LIST OF ALGORITHMS

REFERENCES

[ABC19] Ameer M.S. Abdelhadi, Christos-Savvas Bouganis, and George A. Constantinides. "Accelerated Approximate Nearest Neighbors Search Through Hierarchical Product Quantization." In: *2019 International Conference on Field-Programmable Technology (ICFPT)*. 2019, pp. 90–98. DOI: 10.1109/ICFPT47387.2019.00019.

[ABG06] Esma Aïmeur, Gilles Brassard, and Sébastien Gambs. "Machine Learning in a Quantum World." In: *Advances in Artificial Intelligence (Canadian AI 2006)*. Springer, 2006, pp. 431–442.

[ABG07] Esma Aïmeur, Gilles Brassard, and Sébastien Gambs. "Quantum Clustering Algorithms." In: *Proceedings of the 24th International Conference on Machine Learning (ICML)*. 2007, pp. 1–8. DOI: 10.1145/1273496.1273498.

[ACa21] Katherine Arthur, Nathan Chancellor, and et al. "Balanced k-Means Clustering with a Quantum Annealer." In: *Quantum Information Processing* 20.127 (2021).

[AMS08] P.-A. Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton, NJ: Princeton University Press, 2008.

[AMS09] P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, 2009. ISBN: 978-0-691-13238-3.

[Aur91] Franz Aurenhammer. "Voronoi diagrams—a survey of a fundamental geometric data structure." In: *ACM Comput. Surv.* 23.3 (Sept. 1991), 345–405. ISSN: 0360-0300. DOI: 10.1145/116873.116880. URL: https://doi.org/10.1145/116873.116880.

[AV07] David Arthur and Sergei Vassilvitskii. "k-Means++: The Advantages of Careful Seeding." In: *Proceedings of the 18th Annual ACM–SIAM Symposium on Discrete Algorithms*. 2007, pp. 1027–1035.

[BB84] Charles H. Bennett and Gilles Brassard. "Quantum Cryptography: Public Key Distribution and Coin Tossing." In: *Proceedings of the IEEE International Conference on Computers, Systems and Signal Processing*. Bangalore, India, 1984, pp. 175–179.

[BBK01] Christian Böhm, Stefan Berchtold, and Daniel A. Keim. "Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases." In: *ACM Comput. Surv.* 33.3 (Sept. 2001), 322–373. ISSN: 0360-0300. DOI: 10.1145/502807.502809. URL: https://doi.org/10.1145/502807.502809.

[Bey+99] Kevin Beyer et al. "When Is "Nearest Neighbor" Meaningful?" In: *Database Theory — ICDT'99*. Ed. by Catriel Beeri and Peter Buneman. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 217–235. ISBN: 978-3-540-49257-3.

[Bha43] Anil Bhattacharyya. “On a Measure of Divergence between Two Statistical Populations Defined by Their Probability Distributions.” In: *Bulletin of the Calcutta Mathematical Society* 35 (1943), pp. 99–109.

[Bia+17] Jacob Biamonte et al. “Quantum Machine Learning.” In: *Nature* 549 (2017), pp. 195–202. DOI: 10.1038/nature23474.

[BL14] Artem Babenko and Victor Lempitsky. “Additive Quantization for Extreme Vector Compression.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014.

[Buh+01] Harry Buhrman et al. “Quantum Fingerprinting.” In: *Physical Review Letters* 87.16 (2001), p. 167902. DOI: 10.1103/PhysRevLett.87.167902.

[Das+07] Abhinandan S. Das et al. “Google news personalization: scalable online collaborative filtering.” In: *Proceedings of the 16th International Conference on World Wide Web*. WWW ’07. Banff, Alberta, Canada: Association for Computing Machinery, 2007, 271–280. ISBN: 9781595936547. DOI: 10.1145/1242572.1242610. URL: https://doi.org/10.1145/1242572.1242610.

[DH99] Christoph Dürr and Peter Høyer. “A Quantum Algorithm for Finding the Minimum.” In: *arXiv preprint quant-ph/9607014* (1999).

[Dir58] Paul A. M. Dirac. *The Principles of Quantum Mechanics*. 4th ed. Oxford University Press, 1958.

[EZR19] Benjamin Elizalde, Shuayb Zarar, and Bhiksha Raj. “Cross Modal Audio Search and Retrieval with Joint Embeddings Based on Text and Audio.” In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019, pp. 4095–4099. DOI: 10.1109/ICASSP.2019.8682632.

[Ge+14] Tiezheng Ge et al. “Optimized Product Quantization.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.4 (2014), pp. 744–755. DOI: 10.1109/TPAMI.2013.240.

[GLM08] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. “Quantum Random Access Memory.” In: *Physical Review Letters* 100.16 (2008), p. 160501. DOI: 10.1103/PhysRevLett.100.160501.

[GN98] R.M. Gray and D.L. Neuhoff. “Quantization.” In: *IEEE Transactions on Information Theory* 44.6 (1998), pp. 2325–2383. DOI: 10.1109/18.720541.

[Gro96] Lov K. Grover. “A Fast Quantum Mechanical Algorithm for Database Search.” In: *Proceedings of the 28th Annual ACM Symposium on Theory of Computing* (1996), pp. 212–219. DOI: 10.1145/237814.237866.

[Guo+20] Ruiqi Guo et al. “Accelerating Large-Scale Inference with Anisotropic Vector Quantization.” In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 3887–3896. URL: https://proceedings.mlr.press/v119/guo20h.html.

[HJ12] R. A. Horn and C. R. Johnson. *Matrix Analysis*. 2nd. Cambridge University Press, 2012. ISBN: 978-0-521-84864-2.

[HL04] David R. Hunter and Kenneth Lange. “A Tutorial on MM Algorithms.” In: *The American Statistician* 58.1 (2004), pp. 30–37.

[JDJ17] Jeff Johnson, Matthijs Douze, and Hervé Jégou. "Billion-scale Similarity Search with GPUs." In: *arXiv preprint arXiv:1702.08734* (2017).

[JDS11] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. "Product Quantization for Nearest Neighbor Search." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.1 (2011), pp. 117–128. DOI: 10.1109/TPAMI.2010.57.

[Joz94] Richard Jozsa. "Fidelity for mixed quantum states." In: *Journal of Modern Optics* 41.12 (1994), pp. 2315–2323. DOI: 10.1080/09500349414552171.

[Ker+19] Iordanis Kerenidis et al. "Q-Means: A Quantum Algorithm for Unsupervised Machine Learning." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019, pp. 4136–4146.

[KG09] Brian Kulis and Kristen Grauman. "Kernelized locality-sensitive hashing for scalable image search." In: *2009 IEEE 12th International Conference on Computer Vision*. 2009, pp. 2130–2137. DOI: 10.1109/ICCV.2009.5459466.

[KHK18] Rishabh Kumar, Travis Humble, and Sabre Kais. "Quantum Annealing for Balanced *k*-Means Clustering." In: *Quantum Machine Learning Workshop (QML 2018)*. 2018.

[Llo82] S. Lloyd. "Least squares quantization in PCM." In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137. DOI: 10.1109/TIT.1982.1056489.

[LMR13] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. "Quantum Algorithms for Supervised and Unsupervised Machine Learning." In: *arXiv preprint arXiv:1307.0411* (2013).

[NBS10] Frank Nielsen, Sylvain Boltz, and Olivier Schwander. "Bhattacharyya Clustering with Applications to Mixture Simplifications." In: *Proceedings of the 20th International Conference on Pattern Recognition (ICPR)*. 2010, pp. 1437–1440. DOI: 10.1109/ICPR.2010.355.

[NC10] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. 10th Anniversary Edition. Cambridge University Press, 2010.

[Pen23] Hao Peng. "Quantization to speedup approximate nearest neighbor search." In: *Neural Comput. Appl.* 36.5 (Aug. 2023), 2303–2313. ISSN: 0941-0643. DOI: 10.1007/s00521-023-08920-3. URL: https://doi.org/10.1007/s00521-023-08920-3.

[Roc20] Sebastian Roch. *Spectral theorem and Rayleigh quotient*. https://people.math.wisc.edu/~roch/mdp/roch-mdp-review-spectral-chap5.html. Lecture notes for Math 833 (Modern Discrete Probability), University of Wisconsin–Madison. 2020.

[RS24] Abdul Rashid and Abdul A. Samad. "Generalized Steepest Descent Methods on Riemannian Manifolds and Hilbert Spaces: Convergence Analysis and Stochastic Extensions." In: *arXiv preprint arXiv:2502.19929* (2024). Preprint, available at https://arxiv.org/abs/2502.19929.

[RTJ00] Luca Rigazio, Boris Tsakam, and Jean-Claude Junqua. "Optimal Bhattacharyya Centroid Algorithm for Gaussian Clustering with Applications in Automatic Speech Recognition." In: *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. Vol. 3. 2000, pp. 1599–1602.

[SBP17] Yang Sun, Paresh Babu, and Daniel P. Palomar. "Majorization–Minimization algorithms in signal processing, communications, and machine learning." In: *IEEE Transactions on Signal Processing* 65.3 (2017), pp. 794–816. DOI: 10.1109/TSP.2016.2615381.

[Sho97] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer." In: *SIAM Journal on Computing* 26.5 (1997), pp. 1484–1509. DOI: 10.1137/S0097539795293172.

[Wer18] Dirk Werner. *Funktionalanalysis*. 8., vollständig überarbeitete Auflage. Springer-Lehrbuch. Springer Spektrum, 2018. ISBN: 978-3-662-55406-7. DOI: 10.1007/978-3-662-55407-4. URL: https://doi.org/10.1007/978-3-662-55407-4.

[Wu+22] Teng Wu et al. "A Quantum k-Means Algorithm Based on Manhattan Distance." In: *Quantum Information Processing* 21.309 (2022). DOI: 10.1007/s11128-022-03644-7.

[Xio+20] Lee Xiong et al. *Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval*. 2020. arXiv: 2007.00808 [cs.IR]. URL: https://arxiv.org/abs/2007.00808.

[YR03] Alan L. Yuille and Anand Rangarajan. "The Concave-Convex Procedure (CCCP)." In: *Neural Computation* 15.4 (2003), pp. 915–936.