Optimización de la gestión de memoria en simuladores cuánticos mediante la compresión de datos sin pérdida de precisión

Daniel Adrián González Buendía y Javier Andres Sarmiento Salazar

Trabajo de Grado para Optar al Título de Ingeniero de Sistemas

Director

Luis Alberto Núñez de Villavicencio Martínez

Doctor en Física

Codirector

Gilberto Javier Díaz Toro

Doctor en Ciencias de la Computación

Universidad Industrial de Santander

Facultad De Ingenierías Fisicomecánicas

Escuela De Ingeniería De Sistemas e Informática

Pregrado en Ingeniería de Sistemas

Bucaramanga

2026

Dedicatoria

Este trabajo está dedicado a las personas que acompañaron este proceso de formación personal y profesional.

A nuestras familias.

A nuestras amistades.

A quienes hicieron posible este camino.

A mis padres, María Eugenía Buendía Peña y Joert Adrián González Buendía, por ser el pilar fundamental de mi vida y por acompañarme con esfuerzo, amor y dedicación en cada etapa de mi formación. Gracias por los valores que me inculcaron, por cada sacrificio realizado para brindarme educación y por enseñarme a no rendirme ante las dificultades. Este logro también les pertenece a ustedes, porque ha sido posible gracias a su apoyo constante y a la confianza que siempre depositaron en mí.

A mi hermano, Jaime Andrés Hernandez Buendía, a quien agradezco de manera especial por haber estado a mi lado incluso en los momentos más difíciles, por darme ánimo cuando más lo necesité y por ayudarme a seguir adelante con mis estudios. Su apoyo, compañía y confianza fueron fundamentales para llegar hasta aquí. A mi hermana, Jenny Johanna González Buendía, por ser parte importante de este camino y por su cariño y respaldo a lo largo de esta etapa.

A mi familia, que ha sido mi motivación y fortaleza durante todo este proceso, les dedico este logro con profundo agradecimiento y cariño.

Daniel Adrián González Buendía

A mis padres, Pedro Sarmiento y Claudia Salazar, quienes me han brindado todo el apoyo durante mi vida para poder recorrer este camino. Quienes creyeron en mí y en mis capacidades para poder superarme, alcanzar mis metas y me acompañaron durante todo este proceso. A mi hermano, Cristian Sarmiento, por animarme con sus ocurrencias y estar ahí siempre.

Al amor de mi vida, Silvia Santos, mi compañera incondicional, quien me ha acompañado durante cada día del último trayecto de esta etapa, la razón de mi esfuerzo y la inspiración de mis sueños. Gracias por tu amor eterno, por ser mi refugio, por cada abrazo que me ha dado la fuerza para continuar y no rendirme con nada, por cada palabra de aliento que me ha hecho luchar para conseguirlo incluso cuando yo mismo dudaba de si podría hacerlo. Gracias por creer en mí. Este logro es por y para ti, con mucho amor. Te amo siempre.

Javier Andres Sarmiento Salazar

Tabla de Contenido

Lista de Figuras

Lista de Tablas

Lista de Apéndices

Glosario

Blosc2: Biblioteca de compresión por bloques utilizada en este trabajo para el almacenamiento sin pérdida

HPC: Computación de alto rendimiento (High Performance Computing)

LRU: Least Recently Used, política de reemplazo de caché basada en el uso más reciente

MPI: Message Passing Interface, estándar de comunicación para paralelismo en memoria distribuida

MPS: Matrix Product States, representación tensorial eficiente cuando el entrelazamiento del estado es limitado

OpenMP: API de paralelismo de memoria compartida para programación multihilo

QFT: Transformada Cuántica de Fourier (Quantum Fourier Transform)

ZFP: Biblioteca de compresión para arreglos de punto flotante, empleada aquí como referencia de compresión con pérdida controlada

Resumen

Título: Optimización de la gestión de memoria en simuladores cuánticos mediante la compresión de datos sin pérdida de precisión[1]

Autores: Daniel Adrián González Buendía y Javier Andres Sarmiento Salazar[2]

Palabras clave: Computación cuántica, Simulación cuántica, Gestión de memoria, Compresión de datos

Descripción:

La simulación clásica de circuitos cuánticos sigue siendo un recurso indispensable para diseñar, validar y analizar algoritmos cuánticos. No obstante, en los simuladores de estado completo su alcance práctico queda limitado por el crecimiento exponencial de la memoria requerida para almacenar el vector de estado. En respuesta a esta restricción, el presente trabajo estudia la compresión sin pérdida como mecanismo de gestión de memoria en simuladores cuánticos tipo Schrödinger, con el objetivo de reducir la huella de almacenamiento sin introducir alteraciones numéricas en la simulación.

La propuesta comprende la selección de un simulador base y de una biblioteca de compresión adecuada, el diseño de una arquitectura de almacenamiento por bloques con descompresión selectiva y caché LRU, su implementación en TMFQSfullstate mediante Blosc2 y su evaluación experimental frente a una ejecución de referencia sin compresión y a una estrategia con pérdida controlada basada en ZFP. La validación se realizó con circuitos representativos de comportamientos contrastantes de compresibilidad, en particular Grover y la transformada cuántica de Fourier, para instancias entre 20 y 25 qubits.

Los resultados muestran que la estrategia sin pérdida basada en Blosc logra reducciones significativas de memoria en escenarios favorables, especialmente en Grover, manteniendo coincidencia exacta con la referencia no comprimida y error absoluto máximo nulo en todos los casos evaluados. También evidencian que la eficacia de la compresión depende de la estructura del estado cuántico y del patrón de acceso al vector, por lo que no puede asumirse un beneficio uniforme entre algoritmos. En conjunto, el trabajo demuestra que la compresión sin pérdida es una alternativa viable para ampliar la capacidad práctica de simulación cuando la memoria es la restricción dominante y el estado conserva regularidades aprovechables.

[1]Trabajo de grado

[2]Facultad De Ingenierías Fisicomecánicas. Escuela De Ingeniería De Sistemas e Informática.
Director: Doctor Luis Alberto Núñez de Villavicencio Martínez, Codirector: Doctor Gilberto Javier Díaz Toro

Abstract

Title: Optimization of memory management in quantum simulators through data compression without loss of precision[3]

Authors: Daniel Adrián González Buendía y Javier Andres Sarmiento Salazar[4]

Key words: Quantum Computing, Quantum Simulation, Memory Management, Data Compression

Description:

Classical quantum-circuit simulation remains an essential resource for designing, validating, and analyzing quantum algorithms. In full-state simulators, however, practical scale is limited by the exponential memory required to store the state vector. In response to that restriction, this thesis studies lossless compression as a memory-management mechanism for Schrödinger-style quantum simulators, aiming to reduce storage demands without introducing numerical alterations into the simulation.

The proposed approach includes the selection of a base simulator and a suitable compression library, the design of a block-based storage architecture with selective decompression and LRU caching, its implementation in TMFQSfullstate using Blosc2, and its experimental evaluation against both an uncompressed reference and a controlled-loss strategy based on ZFP. Validation was conducted with representative circuits exhibiting contrasting compressibility behavior, namely Grover's algorithm and the quantum Fourier transform, for instances between 20 and 25 qubits.

The results show that the Blosc-based lossless strategy achieves significant memory reductions in favorable scenarios, especially for Grover, while preserving exact agreement with the uncompressed reference and zero maximum absolute error in all the cases. They also show that compression effectiveness depends on the structure of the quantum state and on state-vector access patterns, so benefits cannot be assumed to be uniform across algorithms. Overall, the thesis demonstrates that lossless compression is a viable alternative for extending practical simulation capacity when memory is the dominant constraint and the state preserves exploitable regularity.

[3]Degree Thesis

[4]Faculty of Physics-Mechanics Engineering. School of Systems Engineering and Computer Science. Advisor: Doctor Luis Alberto Núñez de Villavicencio Martínez, Co-Advisor: Doctor Gilberto Javier Díaz Toro

Introducción

La computación cuántica se ha consolidado como un campo con capacidad para abordar ciertos problemas mediante modelos de cómputo distintos a los de la computación clásica (Nielsen & Chuang, 2010). Sin embargo, el diseño, la depuración y la evaluación de algoritmos cuánticos siguen dependiendo en gran medida de la simulación clásica, que permite trabajar en entornos controlados, reproducibles y comparables antes de llevar los circuitos a hardware real (Díaz et al., 2024; Jones et al., 2019).

Dentro de ese panorama, la simulación de estado completo o tipo Schrödinger ocupa un lugar central por su generalidad: mantiene explícitamente el vector de amplitudes complejas y lo actualiza compuerta por compuerta durante la ejecución del circuito (Jones et al., 2019; Qiskit Development Team, 2025). Esta representación ofrece una referencia detallada y exacta del sistema, pero también impone su principal límite práctico: para n qubits se requieren 2^n amplitudes, de modo que el costo de memoria crece exponencialmente. En doble precisión, ese crecimiento vuelve rápidamente inviable la simulación en hardware convencional; por ejemplo, 40 qubits demandan del orden de 16 TB y, al acercarse a 50 qubits, la exigencia entra en el rango de los petabytes (Häner & Steiger, 2017; Wu et al., 2019).

En este contexto, la compresión sin pérdida de precisión resulta atractiva porque actúa sobre la forma de almacenar el vector de estado sin modificar sus amplitudes. Además, los arreglos científicos de punto flotante pueden contener regularidades aprovechables mediante transformaciones reversibles y compresión por bloques, aun cuando no parezcan compresibles a simple vista (Burtscher & Ratanaworabhan, 2009; Lindstrom & Isenburg, 2006; Masui et al., 2015).

Con base en esta idea, el presente trabajo estudia una estrategia de gestión de memoria para simuladores cuánticos de estado completo implementada sobre TMFQSfullstate (Díaz et al., 2026). La propuesta combina almacenamiento comprimido por bloques, descompresión selectiva y reutilización temporal mediante caché, y se evalúa frente a una referencia no comprimida y a una estrategia con pérdida basada en ZFP. El propósito no es únicamente reducir memoria, sino

determinar en qué condiciones la compresión sin pérdida ofrece una relación favorable entre ahorro de espacio, sobrecarga temporal y exactitud numérica.

El documento se organiza así. Tras el planteamiento del problema y los objetivos, se presentan el marco teórico y el estado del arte; luego se justifican la selección del simulador base y de la biblioteca de compresión; posteriormente se describe la arquitectura propuesta, su implementación en el repositorio intervenido y la evaluación experimental sobre Grover y QFT; finalmente, se sintetizan las conclusiones, alcances y líneas de trabajo futuro.

1 Planteamiento del Problema

En un simulador cuántico tipo Schrödinger, cada incremento en el número de qubits duplica el tamaño del vector de estado. Como consecuencia, el consumo de memoria crece exponencialmente y se convierte en la restricción dominante mucho antes de que otras variables de desempeño se vuelvan críticas (Jones et al., 2019; Wu et al., 2019). El problema no es marginal: limita de forma directa el tamaño de los circuitos que pueden ejecutarse con exactitud en infraestructura clásica de uso académico o de laboratorio.

Las estrategias disponibles para aliviar esta restricción no son equivalentes entre sí. Los métodos basados en redes tensoriales, diagramas de decisión o particionamiento del circuito pueden ser muy efectivos, pero no mantienen necesariamente el mismo modelo de simulación ni la misma relación entre generalidad, tiempo y memoria (Chen et al., 2018; Viamontes et al., 2003; Vidal, 2003). Por su parte, la compresión con pérdida ofrece ahorros importantes, aunque a costa de alterar las amplitudes y de reemplazar la igualdad exacta por métricas de aproximación o fidelidad (Díaz Toro, 2025; Wu et al., 2018).

Esto deja abierto un problema específico: cómo reducir la huella de memoria del vector de estado sin cambiar el modelo de simulación ni introducir error numérico. Resolverlo exige algo más que añadir un compresor al almacenamiento. La utilidad real de la compresión depende de la estructura de los estados generados, de la granularidad con que se particiona el vector y del patrón

de lecturas y escrituras inducido por las compuertas. En otras palabras, una estrategia viable debe integrar compresión, acceso selectivo y reutilización temporal de datos de manera coherente con la dinámica del simulador.

Bajo estas condiciones, el problema central de investigación se formula así: *¿cómo optimizar la gestión de memoria en simuladores cuánticos tipo Schrödinger mediante compresión de datos sin pérdida, de forma que se obtenga un ahorro significativo de memoria con una sobrecarga temporal razonable y se preserve igualdad exacta respecto a una ejecución no comprimida?*

Responder esta pregunta es relevante porque permitiría ampliar la escala práctica de la simulación exacta sin abandonar la representación de estado completo, que sigue siendo una referencia valiosa para validación, comparación y análisis detallado de circuitos cuánticos. Con base en este problema se derivan los objetivos que orientan el desarrollo del trabajo.

2 Objetivos

2.1 Objetivo General

- Diseñar e implementar una estrategia eficiente de gestión de memoria en simuladores cuánticos basada en técnicas de compresión sin pérdida de precisión, con el fin de optimizar el uso de recursos computacionales sin comprometer la exactitud numérica de las simulaciones.

2.2 Objetivos Específicos

- Analizar el impacto del uso de herramientas de compresión de datos sin pérdida de precisión en el consumo de memoria, el tiempo de ejecución y la verificación de igualdad exacta de los resultados respecto a una referencia no comprimida.
- Diseñar una estrategia integral de gestión de memoria basada en compresión sin pérdida de precisión, definiendo el algoritmo, los parámetros de compresión-descompresión y los puntos de integración con el simulador cuántico seleccionado.
- Desarrollar e integrar la estrategia diseñada en el simulador elegido, optimizando la sobrecarga computacional para garantizar un ahorro significativo de memoria sin afectar la exactitud numérica de la simulación.
- Implementar dos algoritmos cuánticos distintos en el simulador cuántico elegido para evaluar el uso de memoria, velocidad y exactitud con y sin compresión de datos.
- Comparar los resultados obtenidos con estrategias tradicionales y métodos que emplean compresión con pérdida de precisión, identificando ventajas y desventajas de cada enfoque.

3 Marco Teórico

3.1 Fundamentos de Información Cuántica

3.1.1 Qubits, Espacio de Estados y Superposición

La unidad básica de información en computación cuántica es el qubit. A diferencia del bit clásico, cuyo valor solo puede ser 0 o 1, un qubit se describe mediante un vector de estado en un espacio de Hilbert generado por los estados base $|0\rangle$ y $|1\rangle$ (Nielsen & Chuang, 2010). Un estado general puede escribirse como

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,,$$

donde α y β son amplitudes complejas que satisfacen la condición de normalización

$$|\alpha|^2 + |\beta|^2 = 1.$$

Esta expresión muestra que el estado cuántico no queda determinado por un valor discreto, sino por una combinación lineal de estados base. Esta propiedad se conoce como superposición y constituye una de las características fundamentales de la información cuántica (Nielsen & Chuang, 2010). En términos observables, al medir el sistema se obtiene uno de los estados base con probabilidades dadas por los módulos cuadrados de las amplitudes.

La formulación anterior puede generalizarse a registros de varios qubits. En ese caso, el espacio de estados total se obtiene mediante el producto tensorial de los espacios individuales, de modo que un sistema de n qubits se describe en un espacio vectorial de dimensión 2^n (Nielsen & Chuang, 2010). Su estado general se expresa como

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i\,|i\rangle\,,$$

donde $|i\rangle$ representa cada estado base de la base computacional y $\alpha_i \in \mathbb{C}$. Por ejemplo, para dos

qubits:

$$|\psi\rangle = \alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle + \alpha_{10}\,|10\rangle + \alpha_{11}\,|11\rangle\,.$$

La superposición adquiere aquí una forma más amplia: el sistema completo puede describirse como una combinación lineal de todos los estados base disponibles. Así, a medida que aumenta el número de qubits, crece también el número de amplitudes necesarias para describir el sistema con exactitud. Esta estructura matemática define el modo en que la información cuántica se representa antes de cualquier implementación computacional.

3.1.2 Entrelazamiento

El entrelazamiento aparece cuando el estado conjunto de un sistema no puede expresarse como el producto tensorial de estados independientes para cada una de sus partes (Nielsen & Chuang, 2010). En estos casos, la información del sistema deja de poder describirse completamente a partir de sus componentes por separado y pasa a estar contenida en el estado global.

Un ejemplo clásico es el estado de Bell

$$\left|\Phi^{+}\right\rangle = \frac{1}{\sqrt{2}}\left(|00\rangle + |11\rangle\right).$$

Este estado no puede factorizarse como $|\psi_1\rangle \otimes |\psi_2\rangle$, por lo que sus componentes no admiten una descripción independiente completa. El entrelazamiento introduce correlaciones no clásicas entre las partes del sistema y constituye uno de los rasgos distintivos de la computación cuántica (Nielsen & Chuang, 2010).

Desde una perspectiva operativa, esta propiedad resulta fundamental porque muchas transformaciones cuánticas generan, modifican o explotan correlaciones de este tipo. En consecuencia, el entrelazamiento no solo es una característica teórica del modelo, sino también una propiedad estructural de gran importancia en la evolución de los estados cuánticos.

3.1.3 Interferencia

La interferencia es la propiedad mediante la cual las amplitudes complejas de un estado cuántico pueden combinarse de forma constructiva o destructiva durante la evolución del sistema (Nielsen & Chuang, 2010). A diferencia de un esquema puramente probabilístico, en un sistema cuántico no se suman directamente probabilidades, sino amplitudes, y solo después se obtienen probabilidades a partir de sus módulos cuadrados.

Esto implica que la fase relativa entre amplitudes desempeña un papel central. Dos contribuciones de magnitud similar pueden reforzarse entre sí o cancelarse parcialmente según su relación de fase. Por esta razón, la evolución cuántica no depende únicamente de cuánto "peso" tenga cada estado base, sino también de cómo se relacionan sus amplitudes complejas.

La interferencia explica cómo ciertas configuraciones del sistema pueden aumentar su probabilidad de observación mientras otras disminuyen, aun cuando todas formen parte de la misma superposición. Junto con la superposición y el entrelazamiento, esta propiedad define la dinámica fundamental con la que la información cuántica es transformada.

3.2 Modelo de Circuitos Cuánticos

3.2.1 Evolución Unitaria y Compuertas Cuánticas

El modelo de circuitos cuánticos representa un algoritmo como una secuencia finita de transformaciones unitarias aplicadas sobre un registro de qubits (Nielsen & Chuang, 2010). Si el estado actual del sistema es $|\psi\rangle$ y se aplica una operación unitaria U, el nuevo estado queda dado por

$$|\psi'\rangle = U\,|\psi\rangle\,.$$

Las compuertas cuánticas son, por tanto, operadores unitarios que transforman el estado preservando su norma. Entre las compuertas de un qubit más utilizadas se encuentran las compuertas de Pauli, la compuerta Hadamard y distintas compuertas de fase.

La compuerta Hadamard se expresa como

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

y actúa sobre los estados base de la siguiente manera:

$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}, \qquad H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}.$$

Esta compuerta es especialmente relevante porque transforma estados base en superposiciones y constituye un bloque elemental en múltiples circuitos.

En términos generales, si un circuito aplica sucesivamente las compuertas $U_1, U_2, \ldots, U_k$, el estado final puede escribirse como

$$\left|\psi_f\right\rangle = U_k \cdots U_2 U_1 |\psi_0\rangle .$$

Esta descripción resalta que el circuito puede interpretarse como una composición ordenada de transformaciones lineales sobre el vector de estado.

3.2.2 Compuertas Controladas y Operaciones sobre Varios Qubits

Además de las operaciones de un solo qubit, los circuitos cuánticos utilizan compuertas que actúan sobre más de un qubit de manera conjunta. Una de las más conocidas es la compuerta CNOT, que aplica una negación controlada sobre un qubit objetivo dependiendo del valor del qubit de control (Nielsen & Chuang, 2010).

Las compuertas controladas permiten construir correlaciones entre qubits y son fundamentales para la generación de entrelazamiento. Por ejemplo, si se parte de $|00\rangle$, se aplica una compuerta

Hadamard al primer qubit y después una CNOT controlada por ese mismo qubit, el resultado es

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}},$$

que corresponde a un estado entrelazado.

En el modelo de circuitos, estas compuertas amplían la capacidad expresiva del sistema al introducir dependencias entre subsistemas. Su efecto puede describirse algebraicamente mediante matrices unitarias de mayor dimensión, aunque en la práctica suelen implementarse mediante reglas de actualización local sobre el vector de estado.

3.3 Del Modelo Cuántico a su Representación Clásica

3.3.1 El Vector de Estado como Representación Computacional

La forma más directa de representar un sistema cuántico en una computadora clásica consiste en almacenar explícitamente su vector de estado. En este enfoque, el sistema de n qubits se materializa como una secuencia de 2^n amplitudes complejas:

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle \longrightarrow [\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_{2^n-1}].$$

Cada elemento del arreglo representa una amplitud del sistema, mientras que su posición identifica el estado base asociado. Esta equivalencia convierte la formulación matemática del estado cuántico en una estructura de datos concreta sobre la cual pueden aplicarse operaciones numéricas.

El uso explícito del vector de estado caracteriza a la simulación de estado completo, también conocida como simulación tipo Schrödinger (Jones et al., 2019). En ella, el sistema se representa manteniendo todas sus amplitudes y actualizándolas conforme se aplican las compuertas del circuito. La principal ventaja de esta aproximación es su correspondencia directa con la formulación matemática del modelo cuántico.

Existen otros enfoques de simulación clásica, como los basados en redes tensoriales,

diagramas de decisión o métodos de suma sobre caminos (Chen et al., 2018; Viamontes et al., 2003; Vidal, 2003). De manera general, estos métodos buscan representar la evolución cuántica mediante estructuras alternativas que explotan diferentes formas de organización algebraica o combinatoria. Sin embargo, la representación explícita por vector de estado ofrece el marco más directo para comprender cómo se traduce un sistema cuántico a memoria clásica y cómo se implementa su evolución sobre arreglos numéricos. Esta distinción también es importante porque no todas las estrategias para reducir memoria cambian la representación matemática del estado: algunas, como la que se estudia en este trabajo, mantienen el formalismo Schrödinger y actúan únicamente sobre la manera de almacenar los datos.

3.3.2 Aplicación de Compuertas sobre el Arreglo de Amplitudes

Una vez que el estado cuántico se representa como un arreglo lineal, la aplicación de compuertas puede entenderse como una secuencia de operaciones numéricas sobre subconjuntos específicos de amplitudes.

En una compuerta de un solo qubit, la transformación no afecta una amplitud aislada, sino pares de amplitudes cuyos índices difieren en el bit asociado al qubit objetivo. En un sistema de tres qubits puede tomarse la convención $(q_2 q_1 q_0)$, donde q_2 es el bit más significativo y q_0 el menos significativo. Con esta convención, la Tabla 1 muestra cómo cambian los pares afectados según la posición del qubit objetivo.

Tabla 1

Pares de amplitudes afectados por una compuerta de un solo qubit en un sistema de 3 qubits, mostrando qué bits permanecen fijos y cuál varía según el qubit objetivo.

Objetivo q_0 (se mantienen fijos q_2 y q_1)		
Bits fijos (q_2q_1)	Estados relacionados	Par afectado
00	$000 \leftrightarrow 001$	(α_0, α_1)
01	$010 \leftrightarrow 011$	(α_2, α_3)
10	$100 \leftrightarrow 101$	(α_4, α_5)
11	$110 \leftrightarrow 111$	(α_6, α_7)

Objetivo q_2 (se mantienen fijos q_1 y q_0)		
Bits fijos (q_1q_0)	Estados relacionados	Par afectado
00	$000 \leftrightarrow 100$	(α_0, α_4)
01	$001 \leftrightarrow 101$	(α_1, α_5)
10	$010 \leftrightarrow 110$	(α_2, α_6)
11	$011 \leftrightarrow 111$	(α_3, α_7)

Esto muestra que el patrón de actualización depende de la posición del qubit sobre el cual se aplica la compuerta. De forma análoga, una compuerta de dos qubits modifica grupos más grandes de amplitudes, pero siempre siguiendo una regla determinada por la estructura binaria de los índices.

Desde el punto de vista computacional, esta regularidad introduce patrones de acceso sobre el arreglo. Algunas operaciones actúan sobre posiciones contiguas y otras sobre posiciones separadas por saltos regulares, de modo que la simulación no solo depende del tamaño total del vector, sino también de cómo se distribuyen las lecturas y escrituras en memoria. En este contexto, las nociones de localidad espacial y localidad temporal ayudan a entender por qué ciertas estrategias de particionamiento, acceso selectivo y reutilización temporal pueden resultar convenientes.

Esta interpretación computacional del circuito es importante porque establece el vínculo entre álgebra lineal y acceso a memoria: simular la evolución cuántica equivale, en este marco, a recorrer, leer, combinar y reescribir posiciones concretas de un arreglo de amplitudes complejas.

3.3.3 Consideraciones Básicas de Almacenamiento y Acceso

Si cada amplitud compleja ocupa 16 bytes, el consumo de memoria del vector de estado puede expresarse como

$$M(n) = 2^n \times 16 \text{ bytes}.$$

Esta relación resume el costo directo de almacenar explícitamente el sistema en una simulación de estado completo (Jones et al., 2019; Wu et al., 2019).

Por tanto, la representación clásica del vector de estado involucra dos aspectos complementarios: cuánto espacio se necesita para almacenar las amplitudes y cómo se accede a ellas durante la evolución del circuito. Estos dos elementos son fundamentales para entender la gestión de memoria sobre arreglos numéricos de gran tamaño.

3.4 Gestión de Memoria y Compresión de Datos en Arreglos Numéricos

3.4.1 Gestión de Memoria en Simulación Científica

La gestión de memoria en computación científica comprende las técnicas utilizadas para organizar, almacenar, recuperar y reutilizar estructuras numéricas de gran tamaño durante la ejecución de un programa. En este tipo de aplicaciones, el rendimiento no depende únicamente del costo aritmético de las operaciones, sino también del comportamiento de los datos en memoria.

Cuando una estructura domina el volumen de almacenamiento y concentra buena parte de los accesos, su organización interna se vuelve especialmente importante. En estos casos, aspectos como la granularidad del acceso, la reutilización temporal y la disposición lineal o segmentada de los datos pueden influir de manera significativa en el comportamiento global del sistema.

Desde esta perspectiva, una estructura como el vector de estado puede analizarse no solo como objeto matemático, sino también como carga principal de memoria dentro de una simulación numérica.

3.4.2 Conceptos Generales de Compresión de Datos

La compresión de datos es el proceso mediante el cual una secuencia de información se representa utilizando menos espacio que en su forma original. La posibilidad de comprimir depende de la existencia de redundancia o regularidad aprovechable en los datos.

En términos generales, los métodos de compresión se clasifican en sin pérdida y con pérdida. La compresión sin pérdida permite recuperar exactamente los datos originales después de descomprimir. La compresión con pérdida, en cambio, admite una reconstrucción aproximada a cambio de mayores reducciones de tamaño.

En datos numéricos, la compresibilidad puede venir de repeticiones, regiones homogéneas, distribuciones estructuradas o relaciones locales entre valores cercanos. Sin embargo, los arreglos en punto flotante presentan una estructura binaria que suele ser menos intuitiva que la de datos textuales o discretos, por lo que su compresión requiere técnicas adecuadas a esa naturaleza.

3.4.3 Compresión sin Pérdida en Datos de Punto Flotante

La compresión sin pérdida de arreglos de punto flotante busca reducir el espacio de almacenamiento preservando exactamente cada valor original. Esto implica que, tras la descompresión, tanto la representación numérica como la secuencia binaria asociada a cada dato deben coincidir con las del arreglo inicial.

Aunque los datos en punto flotante pueden parecer poco compresibles a simple vista, en muchos contextos científicos presentan regularidades explotables. Estas regularidades no siempre se manifiestan como repeticiones exactas de valores, sino también como similitudes locales en exponentes, mantisas o agrupaciones de datos con cierta estructura. Compresores como FPC y FPZIP fueron diseñados precisamente para explotar este tipo de regularidades en arreglos de punto flotante sin alterar sus valores (Burtscher & Ratanaworabhan, 2009; Lindstrom & Isenburg, 2006).

Por ello, la compresibilidad de un arreglo numérico depende tanto del contenido como de la forma en que este se organiza antes de aplicar el compresor. Esta observación es importante cuando

se trabaja con arreglos densos de amplitudes complejas, cuya estructura puede variar según el estado representado.

3.4.4 Compresión por Bloques, Descompresión Selectiva y Caché

Una estrategia habitual para manejar arreglos grandes consiste en particionarlos en bloques y comprimir cada bloque de manera independiente. Esta organización permite trabajar de forma más granular sobre los datos y evita la necesidad de comprimir o descomprimir la estructura completa ante cada operación. En propuestas recientes de simulación cuántica con compresión, este tipo de organización se combina con mecanismos de acceso parcial y administración jerárquica de memoria para reducir presión sobre la RAM (Wu et al., 2019; Zhang et al., 2025).

La descompresión selectiva se basa en esa granularidad: solo se reconstruyen temporalmente los bloques necesarios para una operación determinada. De este modo, la compresión puede integrarse con el patrón de acceso del sistema y no únicamente con el almacenamiento final de los datos.

La granularidad del bloque introduce, sin embargo, un compromiso importante. Bloques grandes suelen capturar mejor la redundancia disponible y favorecer mejores tasas de compresión, pero incrementan el costo cuando solo una pequeña fracción de los datos es necesaria. Bloques pequeños permiten accesos más localizados, aunque pueden degradar el rendimiento del compresor y aumentar el peso relativo del metadato y de la administración.

Cuando ciertos bloques son reutilizados con frecuencia en intervalos cortos, resulta útil mantenerlos temporalmente en una caché. Una política frecuente para administrar esta caché es LRU (*Least Recently Used*), que reemplaza primero los bloques menos recientemente utilizados. Esta política se apoya en la localidad temporal y busca reducir el costo de descompresiones repetidas.

Bloques, descompresión selectiva y caché conforman así un conjunto de conceptos que permiten pensar la compresión como parte activa de la gestión de memoria sobre arreglos numéricos de gran tamaño.

3.4.5 Exactitud Numérica e Igualdad Exacta

En aplicaciones científicas, no siempre basta con obtener resultados aproximados. En muchos casos se requiere preservar exactamente los valores almacenados, de modo que cualquier transformación aplicada a los datos sea completamente reversible.

La compresión sin pérdida satisface este requisito mediante la propiedad de igualdad exacta, según la cual los datos recuperados después de descomprimir coinciden completamente con los originales. Este criterio es más fuerte que otras métricas numéricas basadas en error absoluto, error relativo o fidelidad, ya que no admite desviaciones, por pequeñas que sean. Expresado sobre un vector de amplitudes, este requisito puede escribirse como

$$\alpha_i^{(\mathrm{orig})} = \alpha_i^{(\mathrm{rec})} \quad \forall i.$$

En arreglos de amplitudes complejas, esta propiedad significa que tanto la parte real como la parte imaginaria de cada elemento deben recuperarse sin alteración. Como medida auxiliar de discrepancia, puede emplearse el error absoluto máximo

$$e_{\mathrm{abs}} = \max_i \left| \alpha_i^{(\mathrm{orig})} - \alpha_i^{(\mathrm{rec})} \right|,$$

el cual debe ser nulo en una estrategia estrictamente sin pérdida. Desde un punto de vista metodológico, ello permite distinguir claramente entre optimización del almacenamiento y modificación del contenido numérico.

4 Estado del Arte

4.1 Simuladores Cuánticos

La simulación clásica de circuitos cuánticos es esencial para el desarrollo y validación de algoritmos cuánticos. Sin embargo, el principal desafío radica en la representación del estado

cuántico, cuyo tamaño crece exponencialmente con el número de qubits.

Uno de los primeros enfoques para abordar este problema fue la representación mediante *Matrix Product States* (MPS), propuesta por Vidal (2003). Este método permite representar ciertos estados cuánticos de forma eficiente cuando el entrelazamiento es limitado, reduciendo los requisitos de almacenamiento y cómputo. Su principal ventaja es la escalabilidad en circuitos con baja profundidad, permitiendo simular cientos de qubits. No obstante, su limitación radica en que para sistemas altamente entrelazados el costo computacional crece exponencialmente, restringiendo su aplicabilidad a ciertos algoritmos.

A medida que se buscaban métodos más generales, surgió la simulación tipo *Schrödinger*, que almacena el vector de estado completo y lo actualiza en cada operación cuántica. Este enfoque, empleado en simuladores como QuEST y Qiskit Aer (Jones et al., 2019; Qiskit Development Team, 2025), garantiza alta precisión y permite simular cualquier circuito cuántico. Su desventaja principal es la extrema demanda de memoria, la cual crece exponencialmente con el número de qubits: una simulación de 45 qubits ya ha requerido 0.5 petabytes de memoria (Häner & Steiger, 2017), y al extender esta representación de doble precisión a 50 qubits el costo pasa al orden de decenas de petabytes.

Para mitigar el problema de memoria, se desarrollaron métodos basados en partición del circuito y caminos de Feynman. Por ejemplo, Chen y cols. presentaron un algoritmo distribuido capaz de calcular amplitudes de salida y simular circuitos universales de hasta 12×12 qubits con profundidad moderada sin materializar el estado completo (Chen et al., 2018). Sin embargo, aunque reducen el almacenamiento requerido, estos métodos incrementan la complejidad computacional, ya que el número de caminos crece exponencialmente con la profundidad del circuito.

4.2 Compresión de Memoria con Pérdida de Precisión

Dado que la representación exacta de estados cuánticos es ineficiente en términos de memoria, se han explorado técnicas de compresión con pérdida de precisión. Estos enfoques sacrifican una pequeña cantidad de fidelidad numérica a cambio de una reducción significativa en el uso de

memoria, permitiendo la simulación de circuitos más grandes.

Uno de los primeros enfoques en este ámbito fue el de Wu et al. (2018, 2019), quienes introdujeron la compresión adaptativa con error acotado. Utilizando la biblioteca SZ, reportaron reducciones del tamaño del vector de estado de hasta en un factor de 16 y fidelidades superiores al 99.9 % en los casos experimentales estudiados. Su principal ventaja es que permite ampliar el número de qubits simulados sin un incremento proporcional en los requisitos de memoria. Sin embargo, la compresión y descompresión recurrente introduce sobrecarga computacional, afectando la eficiencia temporal de la simulación.

En un esfuerzo por maximizar la escala de la simulación cuántica de estado completo, Zhang et al. (2025) presentaron *BMQSim*, un marco que extiende *SV-Sim* mediante el uso de **compresores con pérdida de precisión y control de error punto a punto** ejecutados directamente en GPU (por ejemplo, variantes de la familia SZ/cuSZ). Su diseño combina:

- Una estrategia de *circuit partitioning* que disminuye la frecuencia de (des)compresión.
- Un *pipeline* solapado que oculta en gran medida el coste de mover y comprimir datos.
- Una gestión de memoria jerárquica (VRAM + SSD mediante GPUDirect Storage) que asigna dinámicamente espacio a los bloques comprimidos.

Con estas técnicas, BMQSim reduce el consumo de memoria en más de un orden de magnitud y mantiene fidelidades superiores al 99 % en los experimentos reportados sin penalizar de forma apreciable el tiempo de simulación. Su eficacia, sin embargo, está condicionada a la disponibilidad de hardware acelerado con GPUs de alto rendimiento y unidades NVMe rápidas, lo que puede restringir su adopción en plataformas de menor capacidad computacional.

Más recientemente, Huffman et al. (2024) exploraron la cuantización de amplitudes como un mecanismo para reducir la precisión numérica sin afectar la fidelidad global del sistema. Demostraron que, bajo las configuraciones evaluadas, representaciones con tan solo 7 bits por amplitud permiten mantener una fidelidad superior al 99 %, lo que sugiere que la precisión completa de punto flotante puede ser innecesaria en muchas simulaciones. Aunque esta técnica ofrece un enfoque simple y

eficiente para reducir el uso de memoria, su aplicabilidad a circuitos de gran escala aún requiere validación empírica.

Dentro de esta familia de técnicas, ZFP constituye una referencia relevante para arreglos numéricos de punto flotante. Su esquema divide el arreglo en bloques pequeños e independientes y transforma los valores de cada bloque a una representación más compacta antes de codificar sus coeficientes según un criterio de tasa, precisión o error tolerado (Lindstrom, 2014). Esta organización permite comprimir y descomprimir fragmentos del arreglo sin depender del resto, una propiedad útil cuando el acceso a los datos es parcial o segmentado. A diferencia de los enfoques sin pérdida, ZFP no busca preservar igualdad exacta bit a bit, sino controlar el compromiso entre reducción de memoria y desviación numérica. Por esta razón, en este trabajo se toma como referencia de compresión con pérdida controlada frente a la estrategia exacta basada en Blosc.

Díaz Toro (2025) propone un esquema de **vector de estado comprimido** que emplea compresores con pérdida de precisión, principalmente *ZFP* en modo *fixed-rate*, complementado por *SZ*, para reducir la huella de memoria del simulador. El autor reporta reducciones del **10–20 %** en casos base y, en configuraciones de mayor escala, ahorros que alcanzan hasta un **75 %** de la memoria requerida. *ZFP* opera sobre bloques independientes de 4^d valores, de modo que la (des)compresión de cada bloque se realiza en completo aislamiento del resto; esto permite solapar dichas operaciones con el cómputo principal y amortiguar la sobrecarga introducida. Aunque la compresión incrementa el tiempo de ejecución, el impacto se compensa al transmitir los fragmentos del vector en formato comprimido dentro de una arquitectura distribuida basada en *MPI*, reduciendo el volumen de comunicación y habilitando simulaciones de hasta 30 qubits en los experimentos reportados. Finalmente, el estudio concluye que esta estrategia de *estado completo comprimido distribuido* resulta más fiable y escalable que la *poda de estados de baja probabilidad*, la cual presenta descensos de fidelidad y sobrecarga adicional que la hacen desaconsejable.

4.3 Compresión de Memoria sin Pérdida de Precisión

La simulación cuántica eficiente requiere reducir el consumo de memoria sin comprometer la fidelidad del estado cuántico. Para ello, se han desarrollado técnicas de compresión sin pérdida de precisión que buscan representar estados y operadores cuánticos de manera más compacta sin alterar sus valores.

Uno de los primeros enfoques en esta dirección fue el uso de diagramas de decisión cuánticos (*Quantum Information Decision Diagrams*, QuIDD), introducidos por Viamontes et al. (2003). Este método permite representar vectores de estado y operadores unitarios aprovechando redundancias estructurales en sus coeficientes, lo que permite una representación más eficiente en memoria. Su ventaja radica en que, para circuitos con alta redundancia, el crecimiento en memoria puede reducirse de exponencial a polinomial. Sin embargo, su eficacia se limita a circuitos con estructura repetitiva, fallando en casos de entrelazamiento complejo.

En un desarrollo posterior, Miller y Thornton (2006) propusieron los *Quantum Multiple-valued Decision Diagrams* (QMDD), los cuales extienden la representación de QuIDD mediante el uso de aristas ponderadas con matrices unitarias. Esto permitió manejar de manera más eficiente circuitos reversibles y operadores cuánticos de mayor tamaño. Aunque estos diagramas optimizan la representación de puertas lógicas y matrices densas, su eficiencia sigue dependiendo de la estructura interna del circuito.

Más recientemente, Jaques y Häner (2022) exploraron la esparsidad del estado cuántico para almacenar solo las amplitudes no nulas, reduciendo significativamente el almacenamiento requerido en algoritmos con exploración limitada del espacio de Hilbert. Su enfoque demostró ser altamente eficiente en problemas de factorización y aritmética cuántica, pero pierde efectividad en circuitos con alta interferencia y entrelazamiento.

En 2023, Vinkhuijzen et al. (2023) introdujeron los *Local Invertible Map Decision Diagrams* (LIMDD), una evolución de los diagramas de decisión que incorpora transformaciones locales para mejorar la representación compacta del estado cuántico. Esta técnica permite manejar estados

estabilizadores y ciertas estructuras altamente entrelazadas que eran difíciles de representar con diagramas de decisión tradicionales. No obstante, su aplicabilidad sigue limitada a casos donde las transformaciones locales puedan ser explotadas para reducir la complejidad estructural.

Estos avances en compresión sin pérdida han permitido extender la capacidad de simulación de circuitos cuánticos en hardware clásico. Sin embargo, conviene distinguir tres familias de soluciones. Unas cambian la representación matemática del estado, como los diagramas de decisión; otras conservan exactitud, pero dependen de estructura especial, por ejemplo esparsidad o simetrías explotables; y otras todavía dejan abierto el problema de reducir memoria dentro del formalismo Schrödinger con almacenamiento explícito del vector de amplitudes.

Esa distinción es precisamente la brecha que motiva esta tesis. El espacio menos cubierto por la literatura revisada no es el de representar el estado de otra manera, sino el de comprimir sin pérdida un vector de estado explícito, manteniendo la semántica del simulador tipo Schrödinger y gestionando la sobrecarga asociada a acceso, caché y actualización de bloques.

4.4 Enfoques de Compresión sin Pérdida para Datos de Punto Flotante

En simulación científica es frecuente trabajar con arreglos extensos de números de punto flotante, cuyo volumen puede convertir el almacenamiento y el movimiento de datos en una restricción de desempeño. En este contexto, la compresión sin pérdida busca reducir el tamaño de la representación binaria sin alterar ningún bit de la información original, de modo que la reconstrucción coincida exactamente con los datos de entrada. Esta exigencia es especialmente relevante en escenarios donde los valores comprimidos participan posteriormente en nuevas etapas de cálculo o sirven como referencia numérica exacta.

A diferencia de la compresión de texto o de datos discretos, la compresión de punto flotante presenta dificultades particulares. Muchos arreglos científicos contienen redundancias asociadas a continuidad espacial, dependencia temporal o regularidades en la representación binaria de signo, exponente y mantisa. Sin embargo, cuando esas regularidades son débiles o no siguen una estructura local simple, la compresión se vuelve más desafiante. Por esta razón, la literatura ha propuesto

distintos enfoques que no actúan todos del mismo modo: algunos se basan en predicción, otros en transformaciones reversibles de la representación binaria y otros en reorganización tipada de los datos antes de aplicar un códec general.

4.4.1 FPC (Burtscher & Ratanaworabhan)

FPC (*Floating Point Compressor*)(Burtscher & Ratanaworabhan, 2009) es un algoritmo de compresión sin pérdida diseñado para secuencias lineales de valores de punto flotante, especialmente en doble precisión. Su principio central consiste en predecir el siguiente valor a partir del historial reciente y comprimir únicamente la diferencia entre el valor real y el valor estimado.

Para ello, FPC emplea dos predictores derivados de esquemas tipo FCM/DFCM, cuyos resultados se comparan con el dato real mediante una operación `XOR`. Cuando la predicción es cercana, la diferencia resultante presenta una cantidad considerable de bytes iniciales nulos, que pueden codificarse de forma compacta. El algoritmo almacena entonces un pequeño código de control junto con los bytes residuales no nulos. De esta manera, la compresión depende directamente de la capacidad del predictor para anticipar la estructura local de la secuencia.

Este enfoque resulta especialmente adecuado cuando existe correlación entre valores adyacentes o cuando la evolución del arreglo mantiene cierta regularidad. En cambio, su efectividad disminuye cuando los datos no exhiben continuidad local suficiente para que la predicción produzca residuos pequeños. Aun así, FPC constituye una referencia importante en compresión sin pérdida de punto flotante porque muestra con claridad la utilidad y las limitaciones de los esquemas predictivos sobre representaciones IEEE 754.

4.4.2 FPZIP (Lindstrom & Isenburg)

FPZIP(Lindstrom & Isenburg, 2006) es un compresor sin pérdida orientado a arreglos de punto flotante en contextos científicos. Su diseño parte de la idea de que muchos conjuntos de datos numéricos presentan continuidad o dependencia local, de modo que pueden describirse con mayor eficiencia a partir de errores de predicción que a partir de los valores originales.

A diferencia de enfoques lineales simples, FPZIP incorpora esquemas de predicción adaptados a la estructura de los datos, lo que le permite aplicarse a arreglos de distintas dimensionalidades. Tras generar una estimación para cada valor, el algoritmo codifica el residuo de predicción mediante un modelo entrópico diseñado para preservar exactitud bit a bit sobre la representación en punto flotante. En este sentido, FPZIP no requiere cuantización ni aproximación intermedia: la compresión sigue siendo estrictamente reversible.

La relevancia de FPZIP radica en que explota de manera más general la estructura local de arreglos científicos y combina predicción con codificación entrópica especializada. Esto lo convierte en un referente importante para datos en los que la redundancia no se manifiesta únicamente entre valores consecutivos, sino también en relaciones espaciales o estructurales más amplias.

4.4.3 Bitshuffle con LZ4 (Masui et al.)

Una estrategia diferente consiste en no predecir los valores, sino transformar su disposición binaria para hacer visibles regularidades que no son evidentes en el orden original. Bajo esta idea, *Bitshuffle*(Masui et al., 2015) actúa como un filtro reversible que reorganiza los bits de una secuencia de valores de punto flotante antes de aplicar un códec de compresión general, usualmente LZ4.

El procedimiento puede entenderse como una transposición bit a bit: si se consideran N valores de m bits, sus representaciones binarias forman una matriz de $N \times m$, y Bitshuffle reordena los datos agrupando los bits según su posición. Esta transformación puede revelar repeticiones o patrones en determinados planos de bits, por ejemplo en exponentes o signos, que de otra forma quedarían dispersos entre los bytes de cada valor. Una vez realizada la transposición, un códec LZ puede aprovechar con mayor eficacia esas secuencias más homogéneas.

Este enfoque no depende de predicciones locales entre valores adyacentes, sino de la posibilidad de exponer regularidades internas de la representación binaria. Por ello, su interés en compresión científica es doble: por un lado, preserva completamente la exactitud; por otro, ofrece una vía distinta para explotar redundancia en datos donde los modelos predictivos no siempre resultan adecuados. En la práctica, Bitshuffle suele entenderse mejor como una transformación

previa al códec que como un compresor autosuficiente.

4.4.4 Transformación de Datos Tipados (TDT)

La *Transformación de Datos Tipados* (TDT)(Jamalidinan & Cheshmi, 2025) representa una línea más reciente de trabajo orientada a reorganizar datos de punto flotante antes de aplicar compresión generalista. A diferencia de Bitshuffle, que opera sobre una transposición fija de bits, TDT busca identificar posiciones de byte con comportamientos estadísticos diferentes y reagruparlas para producir flujos más homogéneos.

La idea central es que, en una representación IEEE 754, no todos los bytes cumplen el mismo papel ni presentan el mismo nivel de entropía. Los bytes asociados al exponente, al signo o a distintas regiones de la mantisa pueden exhibir distribuciones distintas según la naturaleza de los datos. TDT analiza esas diferencias a nivel de bloque y aplica una reorganización reversible basada en características estadísticas, de modo que los bytes con comportamiento semejante queden contiguos antes de ser entregados a un compresor convencional.

Más que un códec independiente, TDT debe entenderse como un preprocesamiento tipado orientado a mejorar la eficiencia de compresores generales sobre datos numéricos. Su relevancia dentro del estado del arte radica en mostrar que la compresión sin pérdida de flotantes no depende únicamente de predicción o de compresión entrópica, sino también de la capacidad de reordenar la representación binaria para hacer más visible su estructura estadística.

Esta línea de trabajo es especialmente pertinente para el problema de esta tesis porque el vector de estado de un simulador Schrödinger se almacena como una secuencia de amplitudes complejas en doble precisión, usualmente intercaladas como pares real/imaginario. En ese formato, la compresibilidad no depende solo de la magnitud de las amplitudes, sino también de patrones binarios en exponentes y mantisas. Por ello, filtros reversibles como *shuffle* o *bitshuffle* resultan relevantes: pueden agrupar bytes o bits homólogos de valores IEEE 754 y exponer regularidades que no son visibles si cada amplitud se trata como un bloque opaco de 16 bytes.

4.5 Otros Métodos de Optimización de Memoria

Además de la compresión sin pérdida, se han desarrollado enfoques alternativos para reducir el consumo de memoria en simuladores cuánticos, incluyendo técnicas basadas en redes tensoriales, poda de estados y optimización HPC.

Uno de los primeros enfoques en esta dirección fue propuesto por Markov y Shi (2008), quienes introdujeron el uso de redes tensoriales para simular circuitos cuánticos mediante contracción optimizada de tensores. En este método, el circuito se representa como un grafo tensorial, y su simulación se optimiza contrayendo secuencias de tensores en un orden eficiente. Esto permite representar ciertos circuitos con memoria subexponencial, aunque la efectividad del método depende de la estructura del entrelazamiento.

Posteriormente, Boixo et al. (2018) propusieron un modelo gráfico para circuitos cuánticos de baja profundidad basado en la eliminación de variables en grafos probabilísticos. Su técnica permitió simular circuitos aleatorios cercanos al régimen de supremacía cuántica en hardware clásico, extendiendo el tamaño de circuitos simulables sin necesidad de almacenar el estado cuántico completo. Sin embargo, este método pierde eficacia conforme aumenta la profundidad del circuito.

En paralelo, Pednault et al. (2017) introdujeron técnicas de diferimiento de contracciones tensoriales en simulaciones HPC. Su propuesta permite manejar circuitos más grandes al intercambiar memoria por cómputo adicional, almacenando resultados intermedios en memoria secundaria y evitando la expansión total del estado cuántico en RAM. Esta estrategia ha sido clave en la simulación de circuitos con estructuras altamente no triviales, aunque su implementación requiere acceso a infraestructuras de supercomputación.

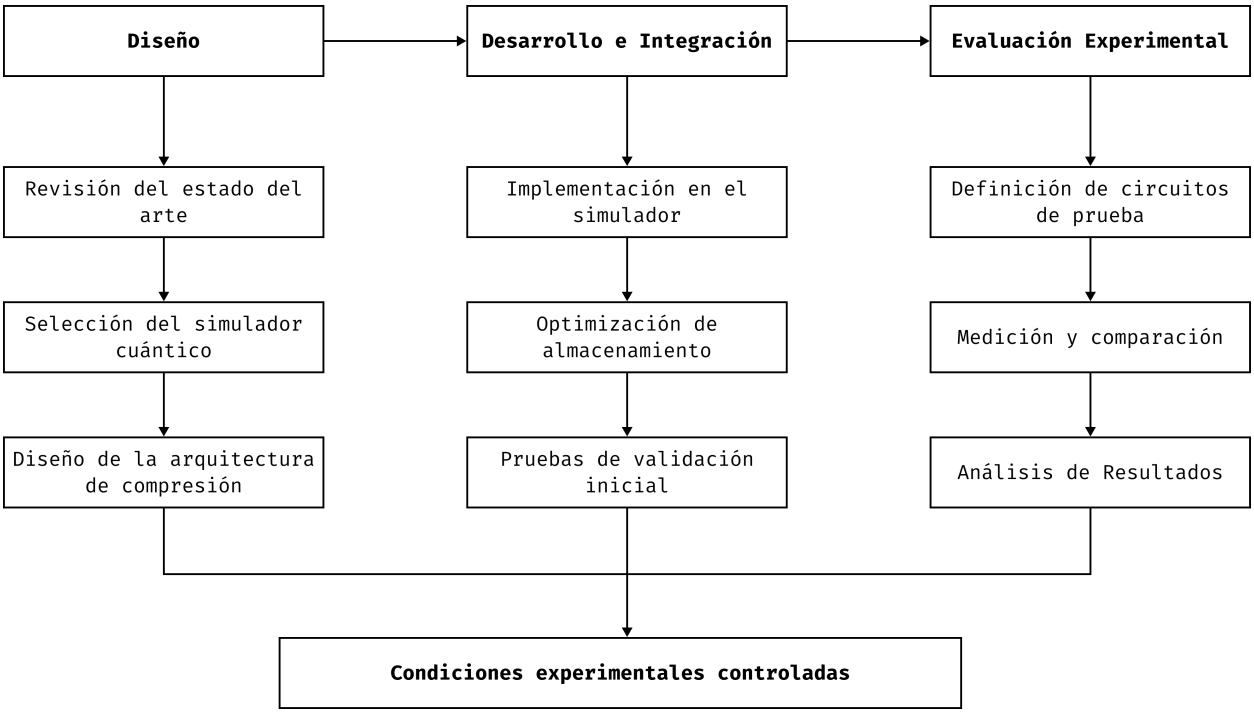
Finalmente, como se analizó previamente, la tesis doctoral de Díaz Toro (2025) introduce técnicas de compresión con pérdida de precisión en esquemas distribuidos de simulación cuántica. Este antecedente resulta especialmente relevante porque consolida el uso de bibliotecas científicas de compresión, vectores de estado y paralelismo distribuido con *MPI* como una línea válida para reducir consumo de memoria en simulación cuántica. Al mismo tiempo, deja visible un espacio de

estudio complementario: evaluar hasta qué punto técnicas de compresión **sin pérdida de precisión** pueden aportar reducciones útiles de memoria sin introducir discrepancias numéricas en el estado simulado.

5 Metodología de la Investigación

El presente trabajo se desarrolló mediante una metodología de investigación aplicada con evaluación experimental. Dado que el problema abordado exigía intervenir un simulador cuántico real, incorporar una estrategia de compresión sin pérdida de precisión y verificar su efecto sobre memoria, tiempo y exactitud numérica, la metodología se estructuró como un proceso de diseño, implementación y validación comparativa. En consecuencia, el trabajo combinó revisión del estado del arte, selección justificada de herramientas, diseño de una arquitectura de almacenamiento, desarrollo de la solución e instrumentación experimental sobre cargas de trabajo representativas. La Figura 1 resume el proceso general seguido.

Figura 1
Metodología de investigación



5.1 Diseño

La fase de diseño estableció la base conceptual y técnica de la propuesta. En esta etapa se realizó la revisión del estado del arte sobre simulación cuántica de estado completo, compresión de datos de punto flotante y estrategias de optimización de memoria, con énfasis en enfoques sin pérdida de precisión aplicables al vector de estado. Esta revisión permitió identificar las alternativas existentes, sus limitaciones y su pertinencia frente al problema específico de reducir la huella de memoria sin alterar el resultado numérico de la simulación.

A partir de este análisis se definieron los criterios para la selección del simulador cuántico base. Se consideraron aspectos como la apertura y modificabilidad del código fuente, la facilidad de intervención sobre la representación del vector de estado, la compatibilidad con optimizaciones de memoria, el soporte de paralelismo y la posibilidad de instrumentar métricas experimentales. La selección se formalizó mediante una matriz de evaluación ponderada entre las alternativas consideradas.

Finalmente, se definió la arquitectura general de la solución. En esta etapa se establecieron la organización del vector de estado en bloques, los mecanismos de compresión y descompresión selectiva, la estrategia de reutilización temporal de datos y los puntos de integración con el simulador seleccionado. Con ello se delimitó el marco de implementación de la propuesta y se fijaron los supuestos que orientarían su evaluación posterior.

5.2 Implementación e Integración de la Estrategia

La segunda fase correspondió al desarrollo de la estrategia diseñada y su integración en el simulador seleccionado. El objetivo fue materializar la arquitectura propuesta en una implementación funcional capaz de operar sobre el almacenamiento del vector de estado sin modificar el modelo general de simulación.

Para ello se intervino el código fuente del simulador en los componentes asociados a la representación, acceso y actualización del vector de amplitudes, incorporando el esquema de almacenamiento comprimido sin pérdida. Esta integración incluyó la gestión de bloques comprimidos, los procedimientos de recuperación y actualización de datos, y los mecanismos requeridos para su persistencia durante la ejecución del circuito.

De forma complementaria, se optimizó el esquema de acceso al almacenamiento con el fin de controlar la sobrecarga asociada a la compresión y descompresión. Estas adaptaciones buscaron favorecer la reutilización temporal de bloques y reducir accesos innecesarios, de modo que el ahorro de memoria no implicara un deterioro desproporcionado del tiempo de ejecución.

Una vez integrada la estrategia, se realizaron pruebas iniciales sobre circuitos de validación para comprobar la estabilidad del desarrollo y verificar, en el caso de la propuesta sin pérdida, la coincidencia exacta del estado final frente a la referencia no comprimida.

Antes de fijar la campaña principal se realizó además una fase preliminar de ajuste de configuración, separada de la evaluación reportada en resultados. En esa etapa se compararon combinaciones de granularidad de bloque, tamaño de caché y parámetros de compresión sobre ejecuciones representativas, con el propósito de seleccionar una configuración estable que mantuviera

ahorro de memoria positivo cuando existía compresibilidad y, al mismo tiempo, evitara penalizaciones temporales desproporcionadas. Una vez escogidos esos parámetros, se mantuvieron constantes en toda la campaña experimental para preservar la trazabilidad metodológica de la comparación.

5.3 Evaluación Experimental

La evaluación experimental se diseñó para determinar el efecto de la estrategia de compresión sobre el comportamiento del simulador en términos de uso de memoria, costo temporal y exactitud numérica. En particular, esta fase buscó establecer en qué condiciones la compresión sin pérdida constituye una alternativa viable frente al almacenamiento no comprimido y cómo se compara con una estrategia complementaria de compresión con pérdida controlada.

La campaña experimental se planteó sobre cargas de trabajo con comportamientos de compresibilidad contrastantes. Para ello se seleccionaron dos familias de circuitos: la Transformada Cuántica de Fourier (QFT) y el algoritmo de búsqueda de Grover. Esta selección permitió evaluar la propuesta en escenarios con estructuras de estado y patrones de acceso diferentes, evitando que la valoración dependiera de un único tipo de circuito.

La comparación se realizó entre tres escenarios: una ejecución de referencia sin compresión, una ejecución con la estrategia de compresión sin pérdida integrada en el simulador y una ejecución con una estrategia de compresión con pérdida controlada. Bajo esta organización fue posible analizar tanto el ahorro de memoria como la sobrecarga temporal y el comportamiento de la exactitud numérica en cada enfoque.

Las métricas principales de evaluación fueron el tiempo total de ejecución y el uso máximo de memoria durante la simulación. Adicionalmente, la exactitud se evaluó de forma diferenciada según la naturaleza de la estrategia comparada. En la propuesta sin pérdida, el criterio de validación consistió en verificar igualdad exacta entre el estado final comprimido y la referencia no comprimida:

$$\alpha_i^{(\mathrm{ref})} = \alpha_i^{(\mathrm{comp})} \quad \forall i.$$

En la estrategia con pérdida controlada, la discrepancia respecto a la referencia se resumió

mediante el error absoluto máximo:

$$e_{\text{abs}} = \max_i \left| \alpha_i^{(\text{ref})} - \alpha_i^{(\text{comp})} \right|.$$

Con el fin de reducir la dependencia de configuraciones particulares, en Grover se consideraron múltiples instancias por tamaño, mientras que en QFT se evaluó una instancia por familia de entrada y por número de qubits. Finalmente, la exactitud se verificó mediante la exportación y comparación completa del vector de amplitudes frente a la referencia no comprimida. En la estrategia sin pérdida, esta comparación permitió confirmar igualdad exacta amplitud por amplitud; en la estrategia con pérdida, el mismo procedimiento se empleó para calcular el máximo error absoluto. De este modo, la evaluación experimental quedó orientada a establecer los compromisos entre memoria, tiempo y exactitud en las estrategias de almacenamiento consideradas.

6 Selección del Simulador Cuántico Base

6.1 Necesidad de Seleccionar un Simulador Base

La propuesta desarrollada en este trabajo exigió intervenir la representación del vector de estado para integrar una estrategia de compresión sin pérdida de precisión y evaluar su efecto sobre memoria, tiempo y exactitud numérica. En consecuencia, la selección del simulador base constituyó una decisión metodológica central, ya que condicionó la viabilidad de la integración, el alcance de la instrumentación experimental y el esfuerzo técnico requerido para desarrollar la solución.

En los simuladores cuánticos de estado completo tipo Schrödinger, el vector de estado es la estructura principal del cálculo y, a la vez, la fuente dominante del consumo de memoria. Por ello, el simulador seleccionado debía permitir una intervención suficientemente directa sobre esa representación, sin trasladar el problema hacia capas internas difíciles de modificar o poco accesibles para observación experimental. Bajo estas condiciones, se consideraron cinco alternativas representativas: QuEST, Intel-QS, qsim, Quantum++ y TMFQSfullstate. Todas ellas pertenecen al

paradigma de simulación de estado completo, pero difieren en el grado de apertura de su arquitectura, en la organización de su código y en el contexto de escalabilidad para el cual fueron concebidas.

6.2 Criterios de Selección

Los criterios de selección se definieron en función del objetivo metodológico del trabajo. Como la investigación requería modificar la forma de almacenamiento del vector de estado sin alterar la semántica general de la simulación, se priorizaron aquellos factores que incidían de manera más directa en la viabilidad de esa intervención y en la posibilidad de extenderla posteriormente. La Tabla 2 resume los criterios definidos con su descripción y ponderación.

Tabla 2
Criterios usados para la selección del simulador cuántico

Código	Criterio	Descripción	Peso
C1	Intervenibilidad del almacenamiento del vector de estado	Evalúa qué tan fácilmente puede reemplazarse, envolverse o rediseñarse la estructura que almacena las amplitudes complejas. Este criterio recibió la mayor ponderación porque el aporte técnico principal del trabajo consiste en modificar la forma de almacenamiento del estado cuántico.	0.45
C2	Desacoplamiento entre lógica de simulación y backend de memoria	Valora en qué medida la lógica asociada a compuertas, evolución del circuito y operaciones sobre el estado puede mantenerse estable cuando se altera la forma de almacenamiento subyacente. Una arquitectura con mayor separación entre estas responsabilidades favorece una integración con impacto controlado sobre el resto del simulador.	0.35
C3	Proyección de escalabilidad y alineación con trabajo futuro	Considera la presencia de mecanismos de paralelismo, distribución o aceleración útiles en extensiones posteriores del trabajo. Aunque este aspecto no fue el eje de la implementación actual, se incorporó para situar la selección en un contexto realista de simulación cuántica y de posibles desarrollos futuros.	0.20

La ponderación asignada refleja la prioridad de intervenir el almacenamiento del vector de estado y mantener control sobre esa intervención. Por esta razón, los criterios de modificabilidad arquitectónica recibieron mayor peso que la proyección de escalabilidad general.

6.3 Simuladores Considerados

Las alternativas evaluadas corresponden a herramientas pertinentes para simulación cuántica de estado completo, aunque con énfasis distintos en cuanto a rendimiento, portabilidad, facilidad de uso o experimentación.

- **QuEST (Quantum Exact Simulation Toolkit):** simulador de alto rendimiento orientado a vectores de estado y matrices de densidad, con soporte para ejecución multihilo, distribuida y acelerada por GPU. Su diseño privilegia escalabilidad y desempeño en arquitecturas heterogéneas (Jones et al., 2019; QuEST-Kit contributors, 2026).

- **Intel-QS (Intel Quantum Simulator):** simulador de circuitos cuánticos basado en representación completa del estado, concebido para aprovechar arquitecturas multinúcleo y multinodo. Su enfoque está orientado a entornos HPC y a la eficiencia en ejecución paralela (Intel Corporation, 2026).

- **qsim:** biblioteca de simulación de estado completo integrada al ecosistema de Cirq, optimizada para calcular el estado final de circuitos cuánticos. Su organización del código distingue componentes para simulación y manejo del vector de estado, con énfasis en ejecución eficiente (Google Quantum AI, 2025; Quantumlib contributors, 2025).

- **Quantum++:** biblioteca general en C++ para simulación cuántica, distribuida como librería *header-only* y diseñada con énfasis en portabilidad, facilidad de uso y rendimiento. Su arquitectura resulta adecuada para prototipado y experimentación ligera (Gheorghiu, 2018).

- **TMFQSfullstate:** prototipo de simulador desarrollado en C++ con foco en estudiar estrategias de representación y consumo de memoria sobre simulación de estado completo. Su diseño

favorece la intervención directa sobre la representación del estado y la exploración de variantes relacionadas con gestión de memoria y paralelismo (Díaz et al., 2024, 2025, 2026).

En conjunto, las alternativas cubren dos perfiles complementarios. QuEST, Intel-QS y qsim representan herramientas con fuerte orientación a rendimiento y ejecución eficiente. Quantum++ y TMFQSfullstate, en cambio, ofrecen condiciones más favorables para intervención directa sobre el código y experimentación sobre la representación del estado.

6.4 Matriz de Evaluación Ponderada

Con base en los criterios anteriores, se aplicó una matriz de decisión ponderada en una escala discreta de 1 a 5, donde 5 representa el mayor grado de adecuación al objetivo del trabajo y 1 el menor. Los puntajes corresponden a una valoración relativa respecto a la posibilidad de integrar y evaluar una estrategia de compresión sin pérdida sobre el vector de estado.

La asignación de valores se estableció a partir de dos fuentes complementarias de evidencia: la documentación oficial y literatura técnica asociada a cada proyecto, y una revisión directa del código disponible en los repositorios o artefactos accesibles de cada simulador. La documentación permitió identificar el propósito de diseño, el modelo de ejecución y las capacidades de paralelismo reportadas por cada herramienta, mientras que la inspección del código permitió valorar la localización de la estructura que representa el estado, el nivel de acoplamiento entre esa estructura y la lógica de simulación, y la facilidad de introducir modificaciones sobre el backend de memoria. Bajo esta escala, la evaluación obtenida fue la siguiente:

Tabla 3
Evaluación de alternativas (escala 1–5) y total ponderado

Simulador	C1	C2	C3	Total ponderado
QuEST	2	2	5	2.60
Intel-QS	2	2	5	2.60
qsim	4	4	4	4.00
Quantum++	4	4	2	3.60
TMFQSfullstate	5	5	3	4.60

Los valores consignados en la tabla responden a la forma en que cada alternativa articula flexibilidad arquitectónica y contexto de ejecución. QuEST e Intel-QS obtuvieron la máxima valoración en proyección de escalabilidad debido a su orientación explícita hacia ejecución multinúcleo, distribuida y, en el caso de QuEST, acelerada por GPU (Intel Corporation, 2026; Jones et al., 2019; QuEST-Kit contributors, 2026). Sin embargo, tanto la documentación como la revisión del código muestran una arquitectura más orientada a desempeño sobre infraestructuras HPC que a modificación experimental del backend de memoria, lo que reduce su valoración en los dos primeros criterios.

En qsim, la separación entre componentes de simulación y manejo del *state vector*, visible tanto en su documentación como en la organización del código, ofrece una base más modular para intervenir el almacenamiento con cambios controlados (Google Quantum AI, 2025; Quantumlib contributors, 2025). Por ello, recibió valores altos en intervenibilidad y desacoplamiento, además de una valoración también alta en escalabilidad por sus variantes orientadas a ejecución eficiente.

Quantum++ obtuvo una valoración alta en los dos primeros criterios por su diseño como biblioteca general en C++ de estructura ligera, distribuida como librería *header-only*, lo que favorece la inspección, modificación y extensión del código (Gheorghiu, 2018). No obstante, su proyección de escalabilidad resulta menor que la de herramientas concebidas desde el inicio para contextos HPC o ejecución distribuida, razón por la cual su valoración en el tercer criterio fue inferior.

TMFQSfullstate alcanzó la valoración más alta en intervenibilidad del almacenamiento y

desacoplamiento entre lógica de simulación y backend de memoria. Tanto la literatura, como la revisión directa del código muestran una base de implementación más directa, con una organización favorable para sustituir la representación del estado y mantener control sobre la lógica de operación del simulador (Díaz et al., 2024, 2025, 2026). Aunque su contexto de escalabilidad es más limitado que el de QuEST o Intel-QS, conserva una proyección suficiente para desarrollos posteriores y, sobre todo, ofrece la mejor adecuación metodológica para integrar la estrategia de compresión propuesta.

6.5 Selección del Simulador

A partir de la matriz de la tabla 3, se seleccionó **TMFQSfullstate** como simulador base para este trabajo.

La elección se sustenta en que ofrece el entorno más adecuado para integrar la estrategia propuesta, observar su efecto sobre el almacenamiento del vector de estado y mantener control experimental sobre la evaluación. Su arquitectura favorece la intervención directa sobre la representación interna del estado y reduce el riesgo técnico de integración.

7 Selección de la Biblioteca de Compresión sin Pérdida

7.1 Necesidad de Seleccionar una Biblioteca de Compresión

Una vez definido el simulador base, la siguiente decisión metodológica consistió en seleccionar la biblioteca de compresión sin pérdida que permitiera materializar la arquitectura propuesta sobre el vector de estado. En este trabajo, la compresión no se plantea como una operación aislada de almacenamiento, sino como parte de un esquema de gestión de memoria basado en particionamiento por bloques, descompresión selectiva y reutilización temporal de datos. Por ello, la biblioteca elegida debía ser compatible con un patrón de acceso repetido de lectura, modificación y escritura sobre fragmentos del estado, y no limitarse a ofrecer una buena tasa de compresión en escenarios secuenciales u orientados a archivo.

Bajo estas condiciones, la selección no debía responder únicamente al ratio de compresión

reportado por una biblioteca, sino a su adecuación como componente de una ruta crítica de simulación numérica. En particular, interesaban bibliotecas utilizables desde C/C++, con operación completamente sin pérdida, soporte eficiente para datos residentes en memoria y comportamiento favorable dentro de una organización por bloques independientes.

7.2 Criterios de Selección

Los criterios de selección se definieron en función del objetivo metodológico del trabajo. Como la investigación requería integrar compresión sin pérdida dentro del backend de memoria del simulador, se priorizaron aquellos factores que incidían de manera más directa en la viabilidad de una arquitectura por bloques, en el costo temporal del acceso a datos comprimidos y en la facilidad de incorporación de la biblioteca en la implementación experimental. La Tabla 4 presenta los criterios definidos con su descripción y ponderación.

Tabla 4
Criterios para la selección de bibliotecas de compresión sin pérdida

Código	Criterio	Descripción	Peso
C1	Adecuación a almacenamiento comprimido por bloques	Evalúa qué tan naturalmente la biblioteca soporta o facilita un esquema de datos particionado en bloques independientes, compatible con compresión y descompresión local sin necesidad de reconstruir el arreglo completo en cada operación. Este criterio recibió la mayor ponderación porque la arquitectura propuesta descansa precisamente en ese patrón de acceso.	0.35
C2	Costo de acceso en la ruta crítica	Valora el comportamiento de la biblioteca cuando la compresión y la descompresión forman parte de accesos repetidos sobre fragmentos del vector de estado. En este contexto, la latencia de recuperación tiene mayor relevancia metodológica que una tasa de compresión aislada obtenida fuera de la dinámica del simulador.	0.30
C3	Facilidad de integración en C/C++ sobre buffers en memoria	Considera la disponibilidad de una interfaz clara para operar directamente sobre buffers residentes en memoria, así como la facilidad de parametrización e incorporación en el backend del simulador. Este criterio recoge el esfuerzo de integración requerido para construir y evaluar la propuesta experimental.	0.20
C4	Afinidad con datos numéricos binarios	Evalúa la cercanía de la biblioteca con arreglos homogéneos de datos binarios o numéricos, en particular de punto flotante, y su compatibilidad con transformaciones reversibles que puedan favorecer la compresibilidad sin alterar la exactitud numérica.	0.15

La ponderación asignada refleja la prioridad de materializar una arquitectura de almacenamiento comprimido por bloques dentro de la ruta de ejecución del simulador. Por esta razón, la adecuación estructural al particionamiento y el costo temporal de acceso recibieron mayor peso que otros atributos más generales de las bibliotecas de compresión.

7.3 Bibliotecas de Compresión sin Pérdida Consideradas

Las alternativas evaluadas corresponden a bibliotecas utilizables desde C/C++ y relevantes para compresión sin pérdida de datos residentes en memoria. Aunque sus objetivos de diseño no son idénticos, todas representan opciones plausibles para integrarse en una arquitectura de almacenamiento comprimido aplicada al vector de estado.

- **Blosc2:** biblioteca orientada a compresión de datos binarios y arreglos numéricos, organizada alrededor de contenedores particionados y de un pipeline de filtros y códecs configurable. Su diseño incorpora filtros como *shuffle* y *bitshuffle*, y permite trabajar con datos en memoria dentro de una organización por fragmentos (Blosc Development Team, 2026).

- **Zstandard (Zstd):** biblioteca generalista de compresión sin pérdida con un amplio rango de compromiso entre velocidad y ratio. Proporciona funciones de compresión y descompresión en memoria y resulta adecuada como códec base cuando el particionamiento y la gestión de bloques son responsabilidad de la aplicación (facebook/zstd contributors, 2026; Zstandard project, 2026).

- **LZ4:** biblioteca de compresión sin pérdida orientada a muy baja latencia, especialmente fuerte en descompresión. Su perfil resulta atractivo cuando el costo temporal del acceso tiene mayor peso que la tasa de compresión (lz4 contributors, 2026).

- **zlib/Deflate:** solución madura y ampliamente adoptada para compresión sin pérdida en memoria y en flujo. Aunque no está especializada en arreglos científicos ni en acceso por bloques, constituye una referencia estable para contrastar bibliotecas más recientes (zlib authors, 2022).

Las cuatro alternativas cubren perfiles distintos. Blosc2 representa una biblioteca con orientación explícita a estructuras particionadas y datos numéricos en memoria; Zstd y LZ4 son bibliotecas generalistas de alto desempeño que pueden emplearse como códecs dentro de una

arquitectura definida externamente; y zlib aporta una referencia clásica, madura y ampliamente conocida.

7.4 Matriz de Evaluación Ponderada

Con base en los criterios anteriores, se aplicó una matriz de evaluación ponderada en una escala discreta de 1 a 5, donde 5 representa el mayor grado de adecuación al objetivo del trabajo y 1 el menor. Los puntajes corresponden a una valoración relativa respecto a la conveniencia de cada biblioteca para integrarse en una arquitectura de almacenamiento comprimido por bloques dentro del simulador seleccionado.

La asignación de valores se apoyó en la documentación oficial y en la literatura técnica asociada a cada biblioteca. Estas fuentes permitieron identificar el propósito de diseño de cada biblioteca, su modelo de uso sobre memoria, la presencia o ausencia de mecanismos internos de particionamiento, y su orientación hacia datos binarios o numéricos.

Tabla 5
Evaluación de bibliotecas de compresión sin pérdida (escala 1–5) y total ponderado

Criterio	Peso	Blosc2	Zstd	LZ4	zlib
C1	0.35	5	3	3	2
C2	0.30	4	4	5	2
C3	0.20	5	4	4	4
C4	0.15	5	2	2	2
Total	**1.00**	**4.75**	3.50	3.65	2.50

La matriz muestra una diferencia clara entre una biblioteca concebida con afinidad hacia datos numéricos particionados y bibliotecas generalistas de compresión sin pérdida. Blosc2 obtuvo la mayor valoración en adecuación a almacenamiento por bloques porque su modelo incorpora contenedores fragmentados, filtros reversibles y operación en memoria dentro de un marco unificado (Blosc Development Team, 2026). Zstd y LZ4 alcanzaron una valoración favorable en costo de acceso e integración debido a sus APIs en memoria y a su perfil de alto desempeño, pero dependen

en mayor medida de que el particionamiento, la política de acceso y la gestión de bloques sean resueltos por la aplicación (facebook/zstd contributors, 2026; lz4 contributors, 2026; Zstandard project, 2026). zlib conserva valor como referencia estable y madura, aunque su perfil resulta menos atractivo para una ruta crítica sensible a latencia y acceso repetido sobre fragmentos del vector de estado (zlib authors, 2022).

Las diferencias más relevantes se concentran en los criterios con mayor peso. Blosc2 sobresale en estructura por bloques e integración porque ya articula un modelo de trabajo alineado con datos binarios y numéricos en memoria. Zstd y LZ4 ofrecen códecs rápidos y ampliamente adoptados, pero trasladan a la aplicación la responsabilidad principal sobre la organización del almacenamiento comprimido. zlib, aunque robusta y muy conocida, resulta menos favorable cuando la prioridad es reducir la sobrecarga temporal del acceso repetido a fragmentos del estado.

7.5 Selección de la Biblioteca de Compresión

A partir de la matriz de la tabla 5, se seleccionó **Blosc2** como biblioteca de compresión sin pérdida para este trabajo.

La elección se sustenta en que Blosc2 ofrece el entorno más adecuado para materializar la arquitectura propuesta de almacenamiento comprimido por bloques. Su diseño está alineado con datos binarios y numéricos en memoria, incorpora filtros reversibles útiles para arreglos de punto flotante y permite combinar compresión y organización por fragmentos dentro de un mismo marco de trabajo (Blosc Development Team, 2026).

Además, Blosc2 permite conservar flexibilidad experimental sin cambiar de biblioteca. Dentro de su marco pueden explorarse distintos filtros, configuraciones y códecs, lo que resulta especialmente valioso en una investigación cuyo interés no es solo comprimir, sino estudiar el compromiso entre ahorro de memoria, sobrecarga temporal y exactitud numérica bajo una arquitectura controlada. En consecuencia, Blosc2 se adopta como la opción más consistente con los objetivos técnicos y metodológicos de este trabajo.

8 Patrones de Compresibilidad en Vectores de Estado de Algoritmos Cuánticos

8.1 Relación entre Estructura del Estado y Compresibilidad

La simulación cuántica de estado completo representa el sistema mediante un vector de amplitudes complejas cuyo tamaño crece como 2^n (siendo n el número de qubits). En este contexto, la *compresibilidad* del vector depende fuertemente de los patrones numéricos que generan los algoritmos: simetrías, valores repetidos, regiones con ceros exactos o distribuciones altamente estructuradas tienden a favorecer la compresión; por el contrario, estados densos con amplitudes variadas y poco redundantes tienden a reducir la efectividad de compresores sin pérdida.

En las subsecciones siguientes se describen patrones típicos de varios algoritmos conocidos y su impacto esperado en compresión de vectores de estado. Esta discusión cumple una función interpretativa y de selección de casos de prueba: no constituye por sí misma la validación experimental del trabajo. La contrastación empírica posterior se concentra en Grover y QFT, que representan dos regímenes estructurales suficientemente distintos para evaluar la estrategia propuesta.

8.2 Búsqueda de Grover: Uniformidad y Baja Entropía

Grover inicia preparando una superposición uniforme sobre $N = 2^n$ estados base, usualmente aplicando compuertas de Hadamard a cada qubit. En esa preparación ideal, todas las amplitudes tienen el mismo valor (magnitud constante), por lo que el vector de estado queda compuesto por un único valor repetido N veces, un caso altamente favorable para compresión sin pérdida (Grover, 1996; Szabłowski, 2021).

Tras cada iteración (oráculo + difusión), la estructura permanece altamente regular: cuando existe un único elemento marcado, el vector puede describirse con dos categorías principales de amplitud (una asociada al estado marcado y otra común al resto de estados no marcados). Esta simetría reduce el número de valores distintos en el vector, lo cual incrementa la redundancia explotable por compresión.

La misma simetría admite una reducción más fuerte cuando el objetivo es simular únicamente la evolución ideal de Grover con un conjunto fijo de estados marcados. Si W denota el conjunto de estados objetivo y $M = |W|$, el estado puede expresarse en el subespacio generado por

$$|w\rangle = \frac{1}{\sqrt{M}} \sum_{x \in W} |x\rangle \,, \qquad |r\rangle = \frac{1}{\sqrt{N-M}} \sum_{x \notin W} |x\rangle \,,$$

de modo que en cada iteración basta actualizar dos parámetros globales en lugar de las N amplitudes individuales. Esta observación no sustituye una estrategia general de almacenamiento, pero sí muestra que la estructura algorítmica de Grover permite optimizaciones adicionales cuando se preserva la igualdad de amplitud dentro de las clases marcada y no marcada. La derivación correspondiente se presenta en el Apéndice A.

De manera empírica, se ha reportado que la compresión permite escalar simulaciones de Grover significativamente más allá de lo que permitiría almacenar el vector sin comprimir. Por ejemplo, Wu y cols. muestran una reducción drástica del requerimiento de memoria para una simulación de Grover de 61 qubits al combinar técnicas de compresión con ejecución en supercomputación (Wu et al., 2019).

8.3 Transformada Cuántica de Fourier: Variación de Fase y Baja Redundancia

La Transformada Cuántica de Fourier (QFT) aplicada a un estado base $|k\rangle$ produce una superposición con magnitudes uniformes pero fases dependientes del índice. En su forma ideal, la amplitud de cada componente $|x\rangle$ puede expresarse como:

$$a_x = \frac{1}{\sqrt{N}} e^{\frac{2\pi i k x}{N}} \,,$$

lo cual implica que, aunque la magnitud sea constante, las partes real e imaginaria varían de forma oscilatoria con x.

Desde la perspectiva de almacenamiento, este comportamiento tiende a generar arreglos *densos* donde la mayoría de entradas son no nulas y muchas de ellas difieren entre sí. En consecuencia,

la redundancia directa (por repetición exacta) suele ser baja para compresión estrictamente sin pérdida. Esta intuición coincide con el tipo de deterioro del ratio reportado por Wu et al. (2018) cuando la evolución del circuito hace más diverso el contenido del vector. Por ello, la QFT constituye un caso razonable para esperar baja compresibilidad al final del circuito, aunque el comportamiento exacto depende del estado de entrada y no debe asumirse idéntico para toda variante del algoritmo.

8.4 Algoritmo de Shor: Periodicidad y Concentración Espectral

Shor combina una etapa de exponenciación modular, que introduce estructura periódica en el registro, con una QFT posterior que redistribuye esa información en el dominio de frecuencias. Desde el punto de vista de compresibilidad, esto sugiere dos regímenes distintos dentro de un mismo algoritmo. Antes de la QFT cabe esperar estados con regularidad aritmética y, según la instancia, con subconjuntos más estructurados que el vector completo; después de la QFT esa regularidad puede traducirse en una distribución más densa y con más variación de fase.

En consecuencia, para una estrategia estrictamente sin pérdida no es prudente afirmar un comportamiento uniforme de Shor. Lo más defendible es señalar que algunas etapas podrían resultar relativamente más favorables que otras: las porciones donde la periodicidad induce patrones repetitivos o ceros exactos serían mejores candidatas para compresión, mientras que las etapas donde la QFT densifica el estado tenderían a reducir esa ventaja.

8.5 Deutsch–Jozsa y Simon: Estructura y Esparsidad

Varios algoritmos tempranos construyen superposiciones altamente estructuradas mediante oráculos e interferencia.

En Deutsch–Jozsa, la interferencia inducida por el oráculo y la transformación final de Hadamard concentra la amplitud sobre un subconjunto reducido de estados base cuando la función pertenece a una de las clases consideradas por el algoritmo. Este tipo de concentración implica esparsidad o repetición exacta de amplitudes, lo que favorece la compresión sin pérdida (Deutsch & Jozsa, 1992; Nielsen & Chuang, 2010).

En Simon, tras medir el registro de salida, el registro de entrada colapsa a una superposición de dos estados relacionados por una máscara secreta. Ese estado intermedio es extremadamente esparso y, por tanto, muy compresible. Posteriormente, la aplicación de compuertas de Hadamard produce una superposición uniforme sobre un subconjunto definido por una restricción lineal, donde reaparecen ceros exactos y amplitudes repetidas (Nielsen & Chuang, 2010; Simon, 1997).

8.6 Circuitos Variacionales y Circuitos Aleatorios: Entre Regularidad y Alta Entropía

En aplicaciones NISQ, algunos circuitos variacionales pueden heredar regularidades de la función objetivo. En esos casos cabe esperar distribuciones de amplitud menos uniformes que las de un circuito completamente aleatorio, por ejemplo con subconjuntos dominantes o con simetrías parciales, lo que en principio favorecería más la compresión que un estado de alta entropía.

En cambio, para circuitos aleatorios profundos lo más razonable es esperar el escenario opuesto: estados densos, con gran diversidad de amplitudes y poca repetición exacta. Bajo esa condición, una estrategia sin pérdida tendría poco margen para explotar redundancia.

8.7 Implicaciones para la Compresión sin Pérdida

Los ejemplos anteriores sugieren una heurística práctica: algoritmos que preservan simetrías fuertes (valores repetidos), esparsidad (ceros exactos) o distribuciones con pocas categorías de amplitud tienden a ser más favorables para compresión sin pérdida. En cambio, transformaciones que producen estados densos y con amplitudes altamente variadas limitan la efectividad de compresores sin pérdida.

Esta relación patrón–compresibilidad es un insumo útil para diseñar experimentos y seleccionar casos de prueba representativos al evaluar técnicas de reducción de memoria en simuladores cuánticos. Resultados experimentales sobre compresión en simulación de circuitos muestran precisamente que, en presencia de estados densos y sin reglas simples de distribución, el ratio puede acercarse a 1 (Wu et al., 2018), mientras que en algoritmos con alta regularidad (como Grover) pueden observarse reducciones de memoria sustanciales (Wu et al., 2019).

9 Diseño de la Arquitectura de Compresión sin Pérdida para Simuladores Tipo Schrödinger

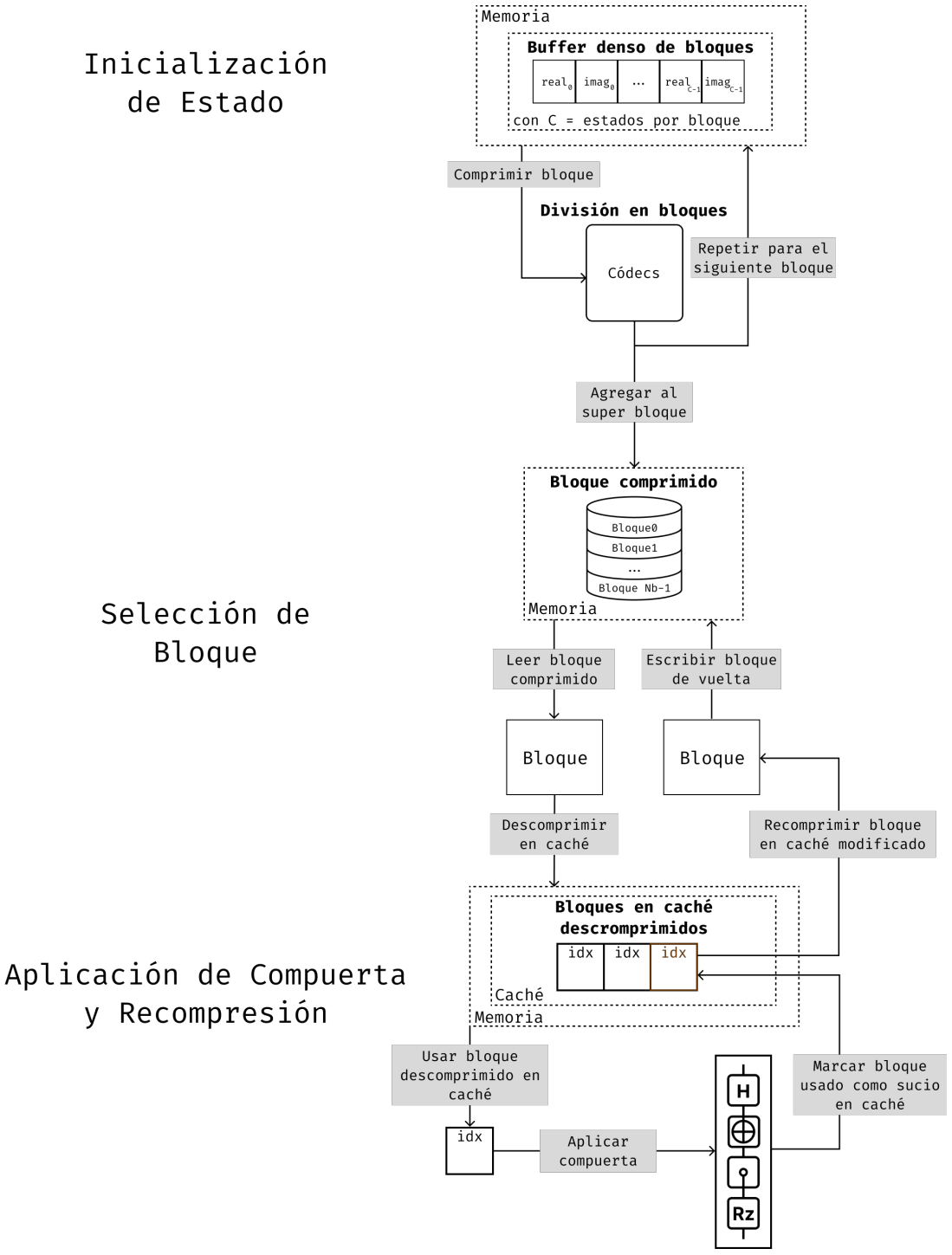
Una vez seleccionado el simulador base y definida la biblioteca de compresión, el siguiente paso consiste en establecer una arquitectura que permita reducir la huella de memoria del vector de estado sin alterar la semántica de la simulación ni comprometer la igualdad exacta de los resultados. Esta exigencia impone una restricción importante: la compresión no puede introducirse como una etapa externa de almacenamiento, sino como parte del mecanismo mediante el cual el simulador representa, accede y actualiza las amplitudes durante la ejecución del circuito.

El análisis del capítulo anterior mostró que la compresibilidad de un vector de estado no es uniforme, sino que depende de la estructura efectiva del estado generado por el algoritmo cuántico. Mientras algunos circuitos producen amplitudes con alta regularidad o redundancia, otros generan distribuciones densas y poco favorables para compresión. Por esta razón, la arquitectura propuesta no asume que todo el vector de estado deba permanecer descomprimido ni que toda operación justifique recomprimir el arreglo completo. En cambio, se adopta un diseño por bloques que busca desacoplar el ahorro de memoria del costo de acceso, permitiendo que la compresión actúe de forma local y bajo demanda.

Bajo esta lógica, la propuesta se organiza alrededor de cuatro decisiones arquitectónicas: particionar el vector de estado en bloques independientes, aplicar descompresión selectiva solo sobre los bloques requeridos por cada operación, mantener una caché temporal de bloques activos y estructurar la compresión como un flujo reversible sobre cada bloque. En conjunto, estas decisiones definen un esquema de almacenamiento comprimido compatible con la dinámica de un simulador cuántico de estado completo y preparan la base para su implementación posterior. La Figura 2 ilustra la organización general de la arquitectura propuesta.

Figura 2

Arquitectura por bloques para la integración de compresión sin pérdida en simuladores cuánticos tipo Schrödinger



9.1 Objetivos de Diseño

El diseño de la arquitectura se orientó por tres objetivos principales. El primero fue preservar la exactitud numérica de la simulación, de modo que la representación comprimida no introdujera diferencias respecto a una ejecución de referencia no comprimida. El segundo consistió en reducir la memoria ocupada por el vector de estado sin exigir su reconstrucción completa ante cada operación. El tercero fue mantener una sobrecarga temporal razonable, de forma que el ahorro espacial no dependiera de una penalización prohibitiva en tiempo de ejecución.

Estos objetivos implican que la arquitectura debía cumplir dos condiciones simultáneamente. Por un lado, toda transformación aplicada al almacenamiento del estado debía ser estrictamente reversible. Por otro, el acceso al vector debía adaptarse al patrón local con que las compuertas actúan sobre las amplitudes, evitando tratamientos globales cuando la actualización solo involucra una fracción del arreglo.

Desde esta perspectiva, el diseño no busca sustituir el modelo de simulación de estado completo, sino modificar su representación física en memoria. El vector de amplitudes se mantiene como referencia lógica del simulador, pero su materialización pasa a depender de una capa intermedia capaz de alternar entre forma comprimida y forma densa según las necesidades de acceso de cada operación. Esta separación entre representación lógica y representación física es la base que articula el resto de la propuesta.

9.2 Representación por Bloques del Vector de Estado

La primera decisión de diseño consiste en particionar el vector de estado en bloques contiguos e independientes. Cada bloque agrupa un subconjunto de amplitudes adyacentes en memoria y se considera la unidad básica de compresión, descompresión, persistencia y reemplazo dentro del sistema. Con esta organización, el vector deja de tratarse como un único arreglo monolítico y pasa a manejarse como una colección de fragmentos sobre los cuales puede actuarse de manera localizada.

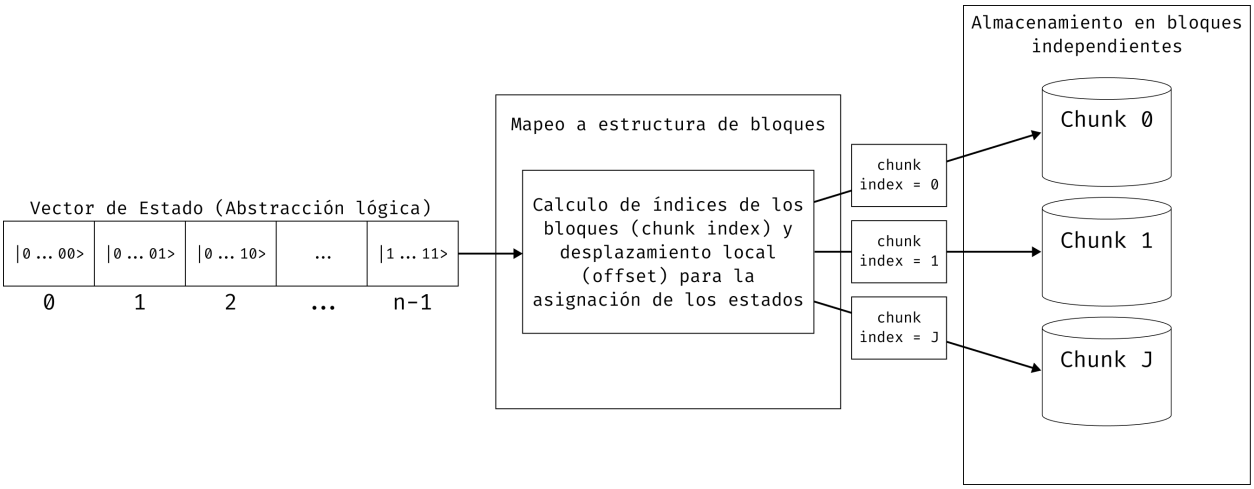
Esta decisión responde a una motivación tanto espacial como operacional. Desde el punto

de vista de memoria, la compresión por bloques permite que partes distintas del vector presenten ratios diferentes según su contenido efectivo. Desde el punto de vista del acceso, evita que una operación que afecta solo una región reducida del estado obligue a descomprimir o recomprimir el arreglo completo. La granularidad del bloque se convierte así en un parámetro central del diseño, ya que determina simultáneamente el potencial de compresión y el costo de recuperar datos para la simulación.

La representación por bloques también favorece una integración más limpia con el simulador. En lugar de alterar la semántica de las operaciones cuánticas, la arquitectura interpone una capa de almacenamiento que resuelve qué bloque contiene cada amplitud, cómo se recupera temporalmente y cuándo debe volver a forma comprimida. Esto conserva el modelo de ejecución del simulador, pero reemplaza la idea de un vector siempre residente en memoria densa por un backend capaz de administrar bloques en distintos estados de materialización. La Figura 3 muestra un ejemplo de esta partición abstracta.

Figura 3

Particionamiento abstracto del vector de estado en bloques independientes



9.3 Descompresión Selectiva y Estrategia de Acceso

Una vez adoptada la representación por bloques, el principio operativo de la arquitectura es que solo deben descomprimirse los bloques necesarios para ejecutar una operación. La descompresión

selectiva constituye, por tanto, la segunda decisión de diseño y el mecanismo que vincula el ahorro de memoria con el patrón real de acceso del simulador.

En un simulador tipo Schrödinger, muchas compuertas actualizan amplitudes cuya relación puede resolverse localmente dentro de una región acotada del vector, mientras que otras inducen interacciones entre posiciones que pertenecen a bloques distintos. La arquitectura debe poder manejar ambos casos sin perder consistencia. Cuando todas las amplitudes requeridas por la operación se encuentran en un mismo bloque, el acceso es enteramente local: el bloque se materializa en memoria densa, se ejecuta la actualización y luego se decide si permanece temporalmente activo o si vuelve a almacenamiento comprimido. Cuando la operación involucra amplitudes distribuidas en bloques distintos, la arquitectura debe adquirir simultáneamente los bloques necesarios, aplicar la actualización y coordinar su posterior permanencia o escritura diferida.

Este enfoque permite que la compresión actúe de forma compatible con la lógica del simulador. El costo de acceso deja de depender del tamaño total del vector y pasa a depender del número de bloques activados por cada compuerta, de la localidad del patrón de acceso y de la frecuencia con la que esos mismos bloques vuelven a ser requeridos. En consecuencia, la efectividad de la arquitectura queda directamente vinculada a la estructura del estado y al tipo de circuito ejecutado, lo cual resulta coherente con el análisis previo de compresibilidad.

9.4 Caché LRU de Bloques Descomprimidos

La descompresión selectiva sería insuficiente si cada bloque tuviera que recomprimirse y recuperarse nuevamente después de cada acceso. Para amortiguar ese costo, la arquitectura incorpora una caché temporal de bloques descomprimidos que actúa como conjunto de trabajo del simulador. Su función es conservar en memoria densa aquellos bloques accedidos recientemente, de modo que las operaciones sucesivas puedan reutilizarlos sin repetir el ciclo completo de recuperación.

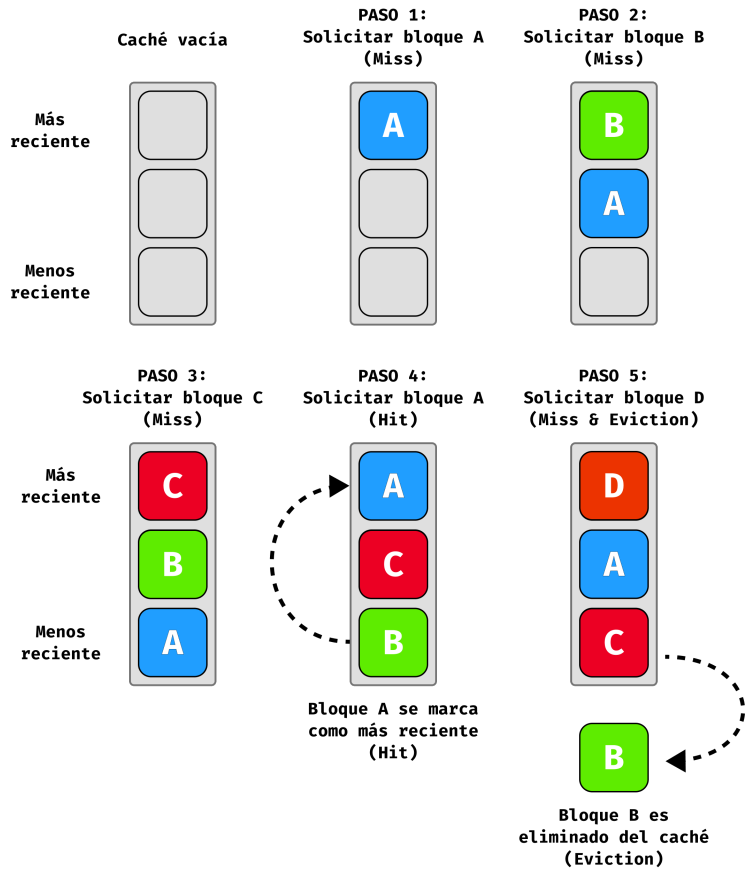
La política de reemplazo adoptada es LRU (*Least Recently Used*). Bajo esta política, cuando la caché alcanza su capacidad y es necesario incorporar un nuevo bloque, se expulsa el bloque cuyo último acceso sea el más antiguo. Esta elección se justifica porque la simulación suele exhibir

localidad temporal: un bloque utilizado en una operación reciente tiene mayor probabilidad de volver a ser requerido en el corto plazo que uno que no ha sido accedido desde varias compuertas atrás.

La caché distingue además entre bloques limpios y bloques modificados. Si un bloque expulsado no ha sufrido cambios desde su recuperación, puede descartarse sin costo adicional. Si el bloque fue modificado, debe recomprimirse antes de abandonar la caché. Esta distinción evita escrituras innecesarias y convierte la caché en un mecanismo no solo de aceleración, sino también de control de sobrecarga. En términos arquitectónicos, la caché LRU cierra el vínculo entre compresión por bloques y acceso selectivo, ya que es el elemento que permite reutilizar materialización densa cuando el patrón de acceso presenta recurrencia. La Figura 4 ilustra la política de reemplazo y los estados posibles de los bloques en la caché.

Figura 4

Política de caché LRU para bloques



9.5 Flujo de Compresión y Descompresión

Sobre cada bloque, la compresión se concibe como un flujo de etapas reversibles. El bloque de amplitudes, representado como un arreglo contiguo en doble precisión, puede someterse primero a una transformación previa sobre su representación binaria, por ejemplo una variante de *shuffle* o *bitshuffle*, con el fin de exponer regularidades que favorezcan la compresión. A continuación, el bloque transformado se entrega al compresor sin pérdida, que genera la representación persistente utilizada por el backend de almacenamiento.

El proceso inverso ocurre únicamente cuando una operación requiere acceder al contenido del bloque. En ese momento, el bloque comprimido se recupera, se descomprime, se invierte la transformación previa y se reconstruye el arreglo denso de amplitudes listo para ser manipulado por la lógica del simulador. La reversibilidad de este flujo garantiza que el bloque obtenido tras la descompresión coincida exactamente con el bloque original antes de ser comprimido, condición indispensable para preservar igualdad exacta frente a la referencia no comprimida.

9.6 Compromisos y Parámetros de Diseño

La arquitectura propuesta presenta dos ventajas principales. La primera es que permite reducir la huella de memoria sin reconstruir de forma permanente el vector completo en su versión densa. La segunda es que introduce parámetros de control claros, como tamaño de bloque, capacidad de caché, política de reemplazo y configuración del compresor, que permiten ajustar la relación entre ahorro espacial y costo temporal según el patrón de acceso observado.

Estas mismas decisiones definen, sin embargo, un espacio de compromiso. Si los bloques son demasiado grandes, aumenta la compresibilidad potencial, pero también el costo de recuperar datos para operaciones locales. Si los bloques son demasiado pequeños, disminuye el costo de acceso, pero puede perderse oportunidad de compresión y aumentar la sobrecarga de metadatos. De forma similar, una caché muy reducida incrementa la tasa de fallos y el número de descompresiones, mientras que una caché demasiado amplia amortigua accesos a costa de reservar más memoria

densa activa.

La arquitectura también hereda una limitación más fundamental: su eficacia depende de la compresibilidad real del estado cuántico generado y del patrón de compuertas que lo recorre. Cuando el estado presenta alta regularidad o redundancia, el diseño puede convertir esa estructura en ahorro efectivo de memoria. En cambio, cuando el circuito genera amplitudes densas y poco compresibles o induce accesos dispersos sobre muchos bloques distintos, la ventaja espacial puede disminuir y la sobrecarga temporal hacerse más visible.

En síntesis, la propuesta delimita una arquitectura de almacenamiento comprimido que no modifica el modelo de simulación de estado completo, pero sí transforma la forma en que el vector de amplitudes reside y circula en memoria. Sobre esta base conceptual, el capítulo siguiente presenta la materialización concreta de estas decisiones en el esquema de almacenamiento comprimido implementado sobre TMFQSfullstate con apoyo de Blosc2.

10 Implementación del Esquema de Almacenamiento Comprimido Basado en Blosc

El desarrollo se apoya en una separación entre la interfaz lógica del registro cuántico y el backend responsable del almacenamiento del estado. En TMFQSfullstate esa separación se materializa en la clase `QuantumRegister`, que conserva la API de operaciones cuánticas, y en la interfaz `IStateBackend`, que abstrae la forma de inicializar, exportar, consultar y mutar el vector de amplitudes.

Esta decisión permite mantener una interacción uniforme con el estado y, al mismo tiempo, introducir una realización comprimida que administra particionamiento por bloques, recuperación bajo demanda, persistencia diferida y reutilización temporal. Así, el simulador conserva su modelo de ejecución compuerta por compuerta, pero delega en un backend especializado la manera concreta en que las amplitudes residen y circulan en memoria.

10.1 Alcance de la Implementación

El objetivo técnico principal no fue rediseñar el simulador en su totalidad, sino intervenir la capa en la que se representan, recuperan y actualizan las amplitudes durante la ejecución.

Bajo este alcance, el desarrollo incluyó tres elementos principales. El primero fue una interfaz común para abstraer el almacenamiento del vector de estado y desacoplar el registro cuántico de una representación específica. El segundo fue un backend comprimido basado en Blosc2, encargado de la persistencia de bloques, la descompresión cuando el acceso lo requiere y la escritura de los bloques modificados. El tercero fue la integración de una estrategia de acceso y actualización compatible con la aplicación de compuertas sobre amplitudes distribuidas en uno o varios bloques.

La implementación buscó, en consecuencia, verificar que la solución propuesta podía insertarse en un simulador cuántico real con control suficiente sobre el uso de memoria, la sobrecarga temporal y la exactitud numérica. Los resultados de esa evaluación se presentan posteriormente en la sección 11. La Tabla 6 resume los componentes principales desarrollados con su responsabilidad y efecto esperado.

Tabla 6
Componentes principales de la implementación basada en Blosc

Componente	Responsabilidad principal	Efecto esperado en tiempo o memoria
`IStateBackend`	Abstraer inicialización, exportación, lectura, escritura y aplicación de compuertas sin exponer la representación física del estado.	Permite sustituir almacenamiento denso por comprimido sin reescribir la lógica del registro.
`BloscState Backend`	Coordinar layout, caché, acceso a amplitudes, aplicación de compuertas y sincronización del estado comprimido.	Reduce memoria persistiendo bloques comprimidos; introduce sobrecarga controlada por accesos y escrituras.
`ChunkLayout`	Mapear cada estado lógico al bloque y desplazamiento local que le corresponden.	Evita materializar el vector completo y habilita acceso selectivo.
`ChunkCache`	Mantener en memoria un conjunto acotado de bloques descomprimidos con política LRU y marcas limpio/sucio.	Disminuye descompresiones repetidas; limita memoria densa temporal.
`BloscCodec`	Crear, leer, actualizar y clonar el super-chunk comprimido de Blosc2.	Encapsula la compresión efectiva y reutiliza contexto/scratch para amortiguar el costo del códec.
`PairKernel Executor` y `GateApply Engine`	Resolver rutas de compuertas intra-bloque, inter-bloque o genéricas.	Reduce accesos innecesarios cuando la geometría del bloque coincide con el patrón de la compuerta.

10.2 Realización del Esquema de Almacenamiento Comprimido con Blosc

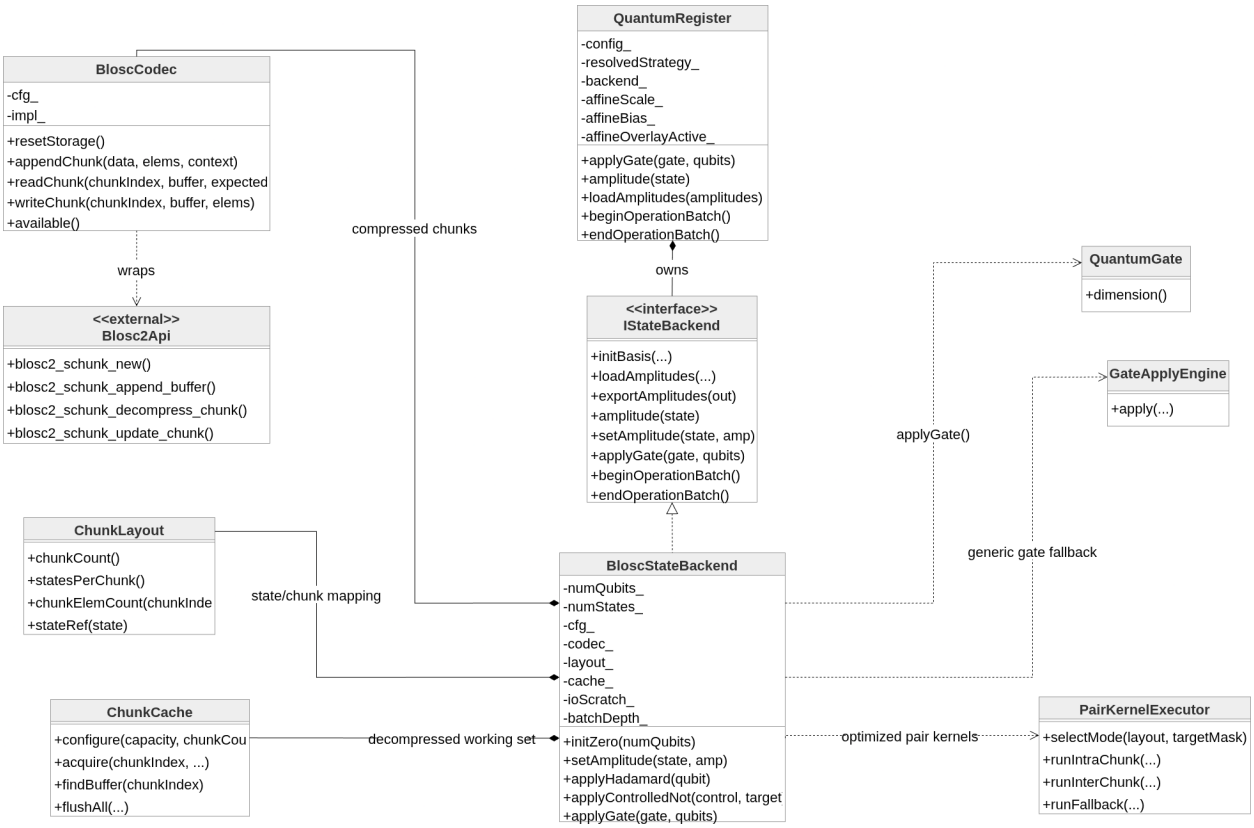
El punto de entrada del desarrollo es la clase `QuantumRegister`, que conserva la interfaz de uso del registro cuántico y delega las operaciones sobre el vector de estado en un backend que satisface la interfaz `IStateBackend`. En el repositorio intervenido, esa interfaz abstrae operaciones de inicialización (`initZero`, `initBasis`, `loadAmplitudes`), operaciones de exportación del estado, acceso elemental (`amplitude`, `setAmplitude`) y aplicación de compuertas, además de dos opera-

ciones opcionales para delimitar lotes mutables (`beginOperationBatch/endOperationBatch`). Esta abstracción fue clave para sustituir el almacenamiento denso sin cambiar la API visible para los algoritmos.

Sobre esa interfaz se implementó `BloscStateBackend`, que constituye la realización concreta del almacenamiento comprimido. La clase mantiene el número de qubits y estados, la configuración resuelta, un `ChunkLayout`, un `ChunkCache`, un `BloscCodec`, un buffer auxiliar de lectura (`ioScratch_`), la profundidad de lote (`batchDepth_`) y un *workspace* reutilizable para compuertas generales. Su responsabilidad central no es "comprimir" en abstracto, sino coordinar el ciclo completo de cada acceso: determinar el bloque implicado, recuperar su versión descomprimida si hace falta, entregar punteros mutables a las amplitudes afectadas, marcar suciedad y decidir cuándo persistir la actualización de vuelta al almacenamiento comprimido.

La compresión efectiva se delega en `BloscCodec`, que encapsula la API de Blosc2 y mantiene el super-chunk comprimido. Este componente valida parámetros como `chunkStates`, `gateCacheSlots`, `clevel` y `nthreads`, crea el contenedor comprimido, anexa bloques en la inicialización y actualiza bloques ya existentes en las escrituras. Además reutiliza un contexto de compresión y un buffer *scratch* para evitar recrear recursos del códec en cada actualización, lo cual reduce parte de la sobrecarga de recompresión.

Figura 5
Diagrama de clases de la implementación del backend comprimido basado en Blosc2.



La figura 5 resume la organización general de esta implementación. En ella se observa la relación entre el registro cuántico, la interfaz de backend, la realización basada en Blosc, los componentes auxiliares de layout y caché, y los mecanismos de ejecución de compuertas.

En conjunto, esta organización permite que la compresión se integre como una responsabilidad propia de la capa de almacenamiento, y no como una operación aislada externa al flujo del simulador. Lo implementado no es un posprocesamiento del vector, sino un backend que mantiene la representación comprimida como estado nativo de ejecución.

10.3 Disposición en Bloques y Mapeo del Vector de Estado

La partición del vector de estado se materializó mediante la clase `ChunkLayout`, encargada de establecer la correspondencia entre índices lógicos del arreglo de amplitudes y bloques físicos

del backend comprimido. En la implementación, esta clase calcula explícitamente: número total de estados, número total de elementos reales almacenados (`2 * totalStates`), cantidad de estados por bloque, elementos por bloque y número de bloques requeridos. Además expone tres consultas decisivas: `chunkElemCount`, para conocer el tamaño real de un bloque; `chunkStateBegin`, para ubicar el primer estado lógico de cada bloque; y `stateRef`, para traducir un índice global a `(chunkIndex, elemOffset)`.

Este mapeo cumple una función central dentro de la implementación. Por una parte, encapsula la granularidad del almacenamiento y evita que otras partes del sistema reconstruyan manualmente la relación entre estados y bloques. Por otra, garantiza un tratamiento correcto del último bloque, cuya cantidad efectiva de amplitudes puede ser menor que la nominal cuando 2^n no es múltiplo exacto de `chunkStates`. Esto es relevante porque el backend opera sobre amplitudes complejas intercaladas `[real, imag, real, imag, ...]`, y por tanto cada estado ocupa exactamente dos elementos dentro del bloque.

En términos operativos, el backend no trabaja con el vector completo como un único buffer comprimido, sino con una secuencia indexada de fragmentos. Cada consulta o modificación pasa primero por `ChunkLayout`: si se solicita la amplitud de un estado, la clase devuelve el bloque que debe cargarse y el desplazamiento local del par real/imaginario; si se recorre el vector por bloques, también permite conocer cuántos elementos deben leerse o escribirse en cada fragmento. Gracias a esta mediación, la recuperación localizada y la escritura posterior pueden resolverse sin materializar el vector completo en memoria densa.

10.4 Caché LRU de Bloques Descomprimidos

La gestión temporal de bloques materializados en memoria se implementó mediante la clase `ChunkCache`, cuya función es conservar un conjunto acotado de bloques descomprimidos recientemente utilizados. Cada entrada de caché contiene el índice del bloque actualmente asociado, un `buffer` con los valores descomprimidos, una marca `dirty` y un contador de uso para la política LRU.

En esta implementación, un bloque se considera *limpio* cuando su contenido descomprimido coincide con la versión persistida en el super-chunk de Blosc2. Se vuelve *sucio* cuando una operación de escritura o una compuerta modifica cualquiera de sus amplitudes y el backend marca explícitamente el `dirty flag` del slot. Esa distinción determina cuándo debe recomprimirse el bloque: los bloques limpios pueden descartarse sin costo adicional, mientras que los bloques sucios deben escribirse de vuelta al backend antes de ser expulsados o al cerrar un lote de operaciones.

La adquisición de bloques se realiza mediante la operación `acquire`. Si el bloque ya está en caché, se reutiliza su buffer y solo se actualiza la marca temporal LRU. Si no está presente y la caché aún tiene capacidad, se crea un nuevo slot y se descomprime el bloque solicitado. Si la caché está llena, se selecciona el bloque menos recientemente utilizado; si ese bloque estaba sucio, se recomprime y persiste antes de liberar el slot para el nuevo bloque. Adicionalmente, el backend permite agrupar varias mutaciones entre `beginOperationBatch` y `endOperationBatch`. En ese intervalo la escritura de bloques sucios se difiere hasta el final del lote, salvo que una expulsión LRU obligue a persistirlos antes.

La utilidad de esta implementación no se reduce a mejorar el rendimiento. La caché establece una forma explícita de administrar la memoria densa temporal del backend y determina qué parte del estado permanece materializada entre una compuerta y la siguiente. Por ello, su comportamiento influye directamente en la cantidad de descompresiones, en la frecuencia de recomposición del super-chunk y en la sobrecarga efectiva del sistema.

10.5 Acceso, Actualización y Ejecución de Compuertas

La aplicación de compuertas sobre el backend comprimido se organizó de acuerdo con el patrón de acceso que cada operación induce sobre las amplitudes. En términos generales, la implementación distingue entre operaciones cuyos pares afectados pueden resolverse dentro de un mismo bloque y operaciones que involucran amplitudes distribuidas en bloques distintos. Esta distinción se implementa de forma explícita en `PairKernelExecutor`, que selecciona tres rutas posibles según la geometría del `ChunkLayout` y la máscara del qubit objetivo: *intra-chunk*,

inter-chunk o *fallback*.

En una compuerta *intra-bloque*, ambos estados del par afectado residen en el mismo fragmento. El backend adquiere un único slot de caché, obtiene dos punteros al mismo buffer descomprimido y aplica allí la transformación. Este es el caso más favorable, porque una sola descompresión permite resolver múltiples pares consecutivos y solo se marca un bloque como sucio, como se ilustra en la Figura 6. En una compuerta *inter-bloque*, los dos estados del par están alineados pero viven en bloques distintos. Entonces el backend adquiere dos slots de caché, uno por bloque, y coordina la actualización cruzada de ambos buffers. Si la capacidad de caché es suficiente, la operación evita recorrer el vector completo y mantiene dos bloques activos mientras procesa todos los pares equivalentes, según el flujo representado en la Figura 7. Cuando ninguna de esas condiciones geométricas se cumple, la ruta *fallback* resuelve estado por estado, reutilizando `acquireAmplitudeRef` para recuperar y actualizar cada amplitud.

La política de compresión y persistencia sigue ese mismo flujo. Un bloque se descomprime cuando `acquireChunk` detecta que no está disponible en la caché. Se recomprime inmediatamente solo en dos situaciones: cuando un slot sucio debe expulsarse por LRU, o cuando finaliza el lote externo y `endOperationBatch` provoca un `flushCache` completo. Si no ocurre ninguna de esas situaciones, la escritura se difiere y el bloque permanece materializado para futuras compuertas, lo que amortiza el costo del códec cuando hay localidad temporal. La Figura 8 esquematiza este flujo de escritura diferida.

Para compuertas frecuentes, el backend además incorpora rutas especializadas. Antes de usar la ruta genérica, `applyGate` detecta matrices equivalentes a Hadamard, Pauli-X, CNOT o compuerta de fase controlada y las redirige a kernels dedicados que operan directamente sobre pares de amplitudes. Solo las compuertas arbitrarias pasan por `GateApplyEngine`, que construye un layout de bloques lógicos de la compuerta y usa callbacks de carga y almacenamiento sobre el backend comprimido. Con ello se reduce el costo del caso común sin perder generalidad.

Desde el punto de vista funcional, esta organización hace posible que la lógica del simulador siga operando sobre amplitudes complejas y compuertas cuánticas del mismo modo que en

una versión no comprimida, mientras la capa de almacenamiento resuelve de forma interna la recuperación, actualización y escritura de los bloques necesarios para cada paso de la simulación.

Figura 6

Flujo de ejecución para compuertas locales sobre un único bloque

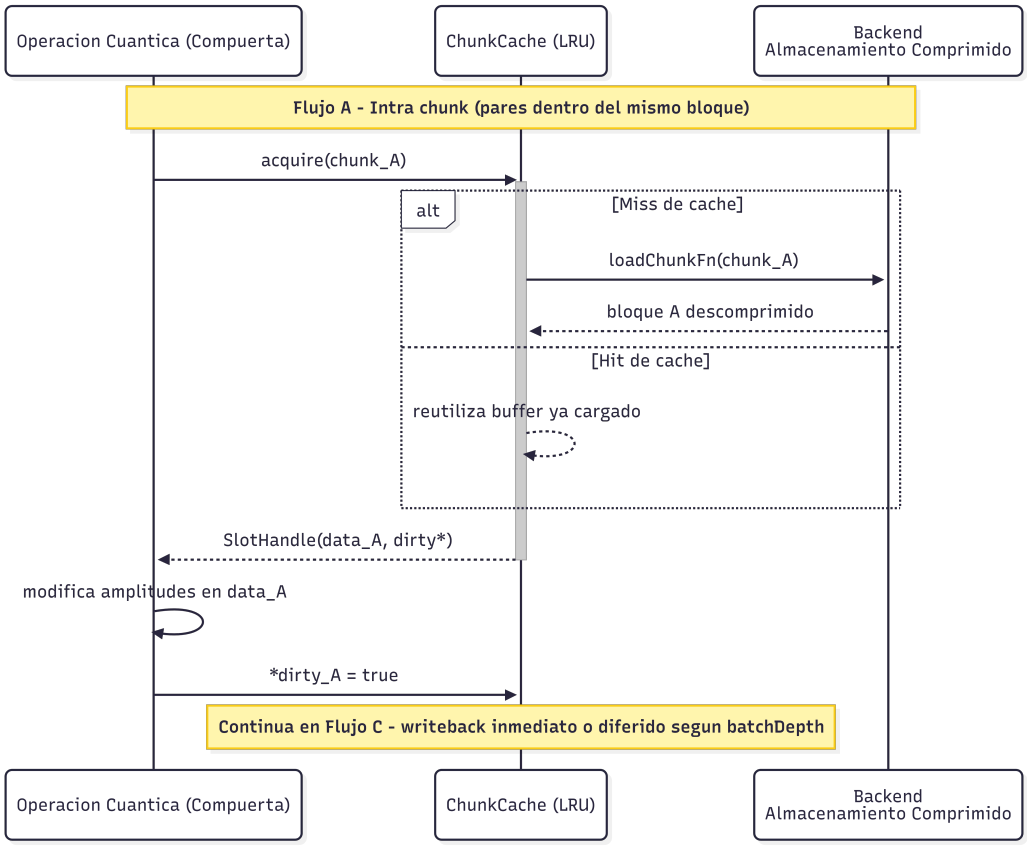


Figura 7

Flujo de ejecución para compuertas que operan entre bloques

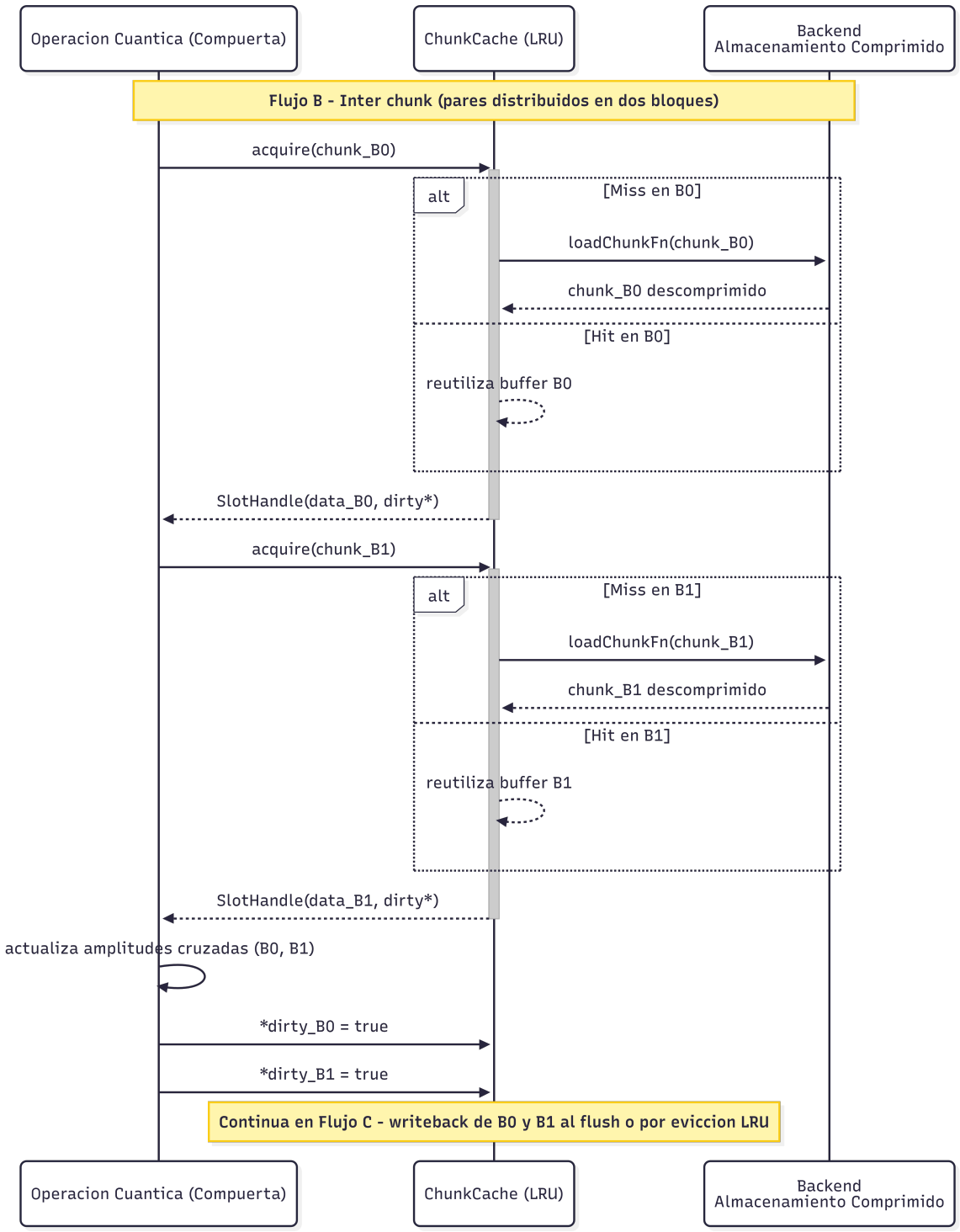
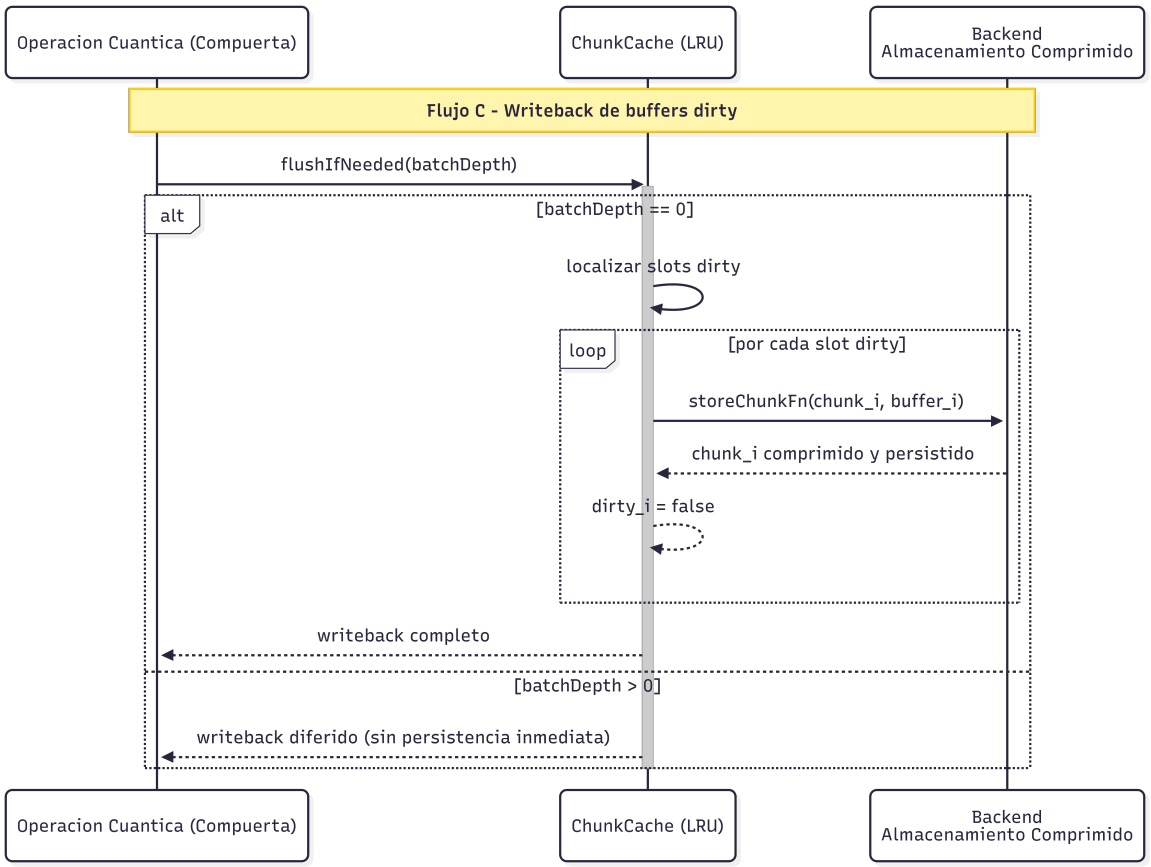


Figura 8
Flujo de persistencia diferida y escritura en backend comprimido



10.6 Optimización Específica para Grover mediante Parametrización Reducida

Además del backend general basado en bloques comprimidos, el desarrollo incorporó una optimización específica para el algoritmo de Grover. Esta variante parte de la observación de que, en ciertos escenarios, la evolución del estado puede describirse mediante una parametrización reducida asociada a la estructura del algoritmo, sin necesidad de operar constantemente sobre el vector completo de amplitudes, su derivación matemática se puede encontrar en el apéndice A.

Su propósito no fue sustituir la implementación general, sino ofrecer una referencia complementaria para contrastar el efecto de una simplificación estructural específica frente al almacenamiento comprimido aplicable de manera general. Mientras el backend basado en Blosc2

conserva una representación explícita del estado y modifica su forma de almacenamiento, la variante reducida aprovecha una regularidad particular de Grover y, por tanto, constituye una optimización especializada.

Esta inclusión resulta útil porque permite distinguir entre dos fuentes distintas de mejora: por un lado, las obtenidas mediante compresión sin pérdida sobre el vector de estado; por otro, las derivadas de una representación más compacta sustentada en la estructura del algoritmo. Sus resultados se presentan más adelante como comparación complementaria y no como reemplazo de la propuesta principal.

En conjunto, la implementación desarrollada organiza el almacenamiento comprimido como una combinación de interfaz, backend, layout de bloques, caché y mecanismos de ejecución de compuertas dentro de TMFQSfullstate. Esta estructura permite integrar Blosc2 en la ruta de simulación sin modificar el modelo de estado completo y prepara la base para la evaluación experimental presentada en el capítulo siguiente.

11 Evaluación Experimental y Resultados

Esta sección presenta el diseño experimental y la discusión de resultados empleada para evaluar la estrategia propuesta de almacenamiento comprimido. La evaluación se formuló como una comparación controlada entre almacenamiento no comprimido, compresión sin pérdida y compresión con pérdida controlada, usando cargas de trabajo con distinta compresibilidad, un mismo entorno de ejecución y criterios uniformes de medición. El objetivo fue examinar de manera conjunta el ahorro de memoria, la sobrecarga temporal y la exactitud numérica, y determinar en qué condiciones la compresión sin pérdida constituye una alternativa práctica dentro del modelo de simulación adoptado.

11.1 Diseño Experimental

11.1.1 Estrategias Comparadas

El experimento se organizó alrededor de tres estrategias de representación del vector de estado: una referencia sin compresión, la propuesta sin pérdida basada en Blosc y una alternativa con pérdida controlada basada en ZFP. Blosc representa la estrategia exacta implementada en el simulador, mientras que ZFP se incluyó como referencia aproximada sobre bloques independientes de valores de punto flotante. Esta elección permitió contrastar, bajo una organización general comparable del almacenamiento, los compromisos entre ahorro de memoria, sobrecarga temporal y alteración numérica. Las tres estrategias se ejecutaron sobre los mismos circuitos, con iguales tamaños de qubits y bajo un procedimiento homogéneo de medición, de modo que las diferencias observadas pudieran atribuirse al esquema de almacenamiento y no a cambios en la carga, en la instrumentación o en el entorno de ejecución.

11.1.2 Selección de Cargas de Trabajo

La evaluación se estructuró sobre dos familias de circuitos con comportamientos contrastantes en términos de compresibilidad: el algoritmo de búsqueda de Grover y la transformada cuántica de Fourier (QFT). Esta selección evitó sesgar la evaluación hacia un único tipo de estado cuántico. Grover representa escenarios en los que pueden mantenerse regularidades aprovechables por la compresión, mientras que QFT permite examinar casos en los que esa estructura se reduce o desaparece. El diseño, por tanto, cubre tanto condiciones favorables para la propuesta como situaciones menos propicias, necesarias para delimitar su alcance real.

Casos Evaluados para Grover. En Grover se estudiaron dos variantes. La primera correspondió a un oráculo con objetivo único y se ejecutó con tres ubicaciones representativas del estado marcado: $N/8$, $N/2$ y $7N/8$, donde $N = 2^n$ es el tamaño del espacio de búsqueda. La segunda correspondió a un oráculo multiobjetivo con tres estados marcados distribuidos en esas mismas regiones. Con esta definición se buscó que los resultados no dependieran de una sola disposición de índices y, al mismo tiempo, observar si la posición de los elementos marcados introducía variaciones apreciables en memoria o tiempo de ejecución.

Casos Evaluados para QFT. En QFT se consideraron dos familias de entrada. La primera fue una superposición periódica, incluida por su estructura regular y repetitiva. La segunda fue una superposición de alta entropía con amplitudes de magnitud uniforme y fases aleatorias, definida como $a_y = e^{i\phi_y}/\sqrt{N}$ con $\phi_y \sim U[0, 2\pi)$ y semilla fija. Este contraste permitió enfrentar la propuesta tanto a estados con redundancia potencialmente explotable como a estados donde dicha redundancia es escasa, mientras que el uso de una semilla fija preservó la reproducibilidad del experimento.

11.1.3 Escala del Experimento

La evaluación se ejecutó para tamaños entre 20 y 25 qubits. Este rango permitió trabajar con instancias suficientemente grandes para que la memoria se convirtiera en una restricción

relevante dentro de la capacidad disponible del entorno experimental, sin perder reproducibilidad ni comprometer la comparabilidad entre estrategias. Elegir varios tamaños dentro de ese intervalo también permitió observar cómo evolucionaban los efectos de la compresión a medida que crecía el vector de estado.

11.1.4 Entorno Experimental e Instrumentación

Todas las ejecuciones se realizaron en una plataforma experimental con Rocky Linux 10.1, procesador AMD Ryzen 7 5700X de 8 núcleos y 16 hilos, y 32 GiB de memoria RAM. La recolección de métricas se efectuó con los scripts de perfilado del repositorio del simulador, que ejecutan los binarios experimentales y registran tiempo transcurrido y memoria máxima mediante `psrecord`. Se recurrió a `psrecord` porque, al operar como una herramienta externa al código instrumentado, permite observar el consumo de recursos del proceso sin introducir lógica adicional en la implementación evaluada. Con ello se buscó reducir la interferencia de la propia medición sobre el experimento y mantener un procedimiento uniforme para todas las estrategias comparadas.

11.1.5 Configuración de Compresión

Las estrategias comprimidas se evaluaron con configuraciones fijas durante toda la campaña experimental, resumidas en la Tabla 7. Esta decisión se adoptó después de una fase preliminar de ajuste, separada de la evaluación principal, en la que se exploraron valores de granularidad de bloque, tamaño de caché y parámetros de compresión sobre ejecuciones representativas. El criterio de selección fue privilegiar configuraciones que mantuvieran ahorro de memoria positivo cuando existía estructura compresible y que, simultáneamente, evitaran una sobrecarga temporal excesiva frente a la referencia. Una vez fijadas, las configuraciones se conservaron sin cambios para impedir que la comparación quedara condicionada por un reajuste particular en cada caso.

En particular, los valores `chunkStates` = 32768 y `gateCacheSlots` = 8 se eligieron porque ofrecieron una granularidad suficiente para amortiguar el costo del códec sin forzar materializaciones demasiado grandes, y porque permitieron retener en memoria de trabajo los bloques

más reutilizados durante la aplicación de compuertas. En Blosc, `clevel` = 1 y `useShuffle` = `true` se fijaron por representar una configuración conservadora, favorable al tiempo de ejecución y coherente con arreglos homogéneos de dobles intercalados. En ZFP, el modo `FixedPrecision` con `precision`=40 se mantuvo como punto de comparación con pérdida controlada, priorizando una discrepancia numérica pequeña sin eliminar por completo el potencial de reducción de memoria. La evaluación de ZFP no se planteó como mecanismo de exactitud, sino como línea base para observar cuánto ahorro adicional podía obtenerse al admitir una desviación numérica acotada.

Tabla 7
Configuraciones de compresión utilizadas en toda la campaña experimental

Estrategia	**Configuración**
Blosc	`chunkStates` = 32768, `gateCacheSlots` = 8, `clevel` = 1, `nthreads` = 4, `compcode` = 1, `useShuffle` = `true`.
ZFP	Modo `FixedPrecision`, `precision` = 40, `chunkStates` = 32768, `gateCacheSlots` = 8, `nthreads` = 4.

11.1.6 Métricas de Evaluación

Las métricas principales del estudio fueron el tiempo total de ejecución y la memoria máxima. Ambas se obtuvieron a partir de las trazas generadas por `psrecord`: el tiempo corresponde al último valor registrado en la columna *Elapsed time*, mientras que la memoria máxima corresponde al mayor valor de la columna *Real (MB)*. A partir de estas medidas se calcularon dos indicadores de lectura directa:

$$\text{Ahorro de memoria (\%)} = 100\left(1 - \frac{M_{\text{comp}}}{M_{\text{ref}}}\right), \qquad \text{Tiempo relativo (x)} = \frac{T_{\text{comp}}}{T_{\text{ref}}}.$$

En estas expresiones, M_{ref} y T_{ref} corresponden a la ejecución con almacenamiento no comprimido del mismo circuito y del mismo número de qubits. Un valor negativo en el ahorro de memoria indica que la estrategia evaluada consumió más memoria que la referencia, lo que permite

identificar con claridad los casos en los que la compresión no alcanza a compensar sus propios costos auxiliares.

11.1.7 Verificación de Exactitud Numérica

La exactitud numérica se evaluó según la naturaleza de cada estrategia. En Blosc, al tratarse de una compresión sin pérdida, la validación consistió en exportar el vector completo de amplitudes de cada ejecución comprimida y compararlo amplitud por amplitud con la ejecución no comprimida de la misma instancia, con el fin de verificar igualdad exacta respecto a la referencia. En ZFP, esa misma comparación completa se empleó para medir el máximo error absoluto, ya que en este caso el interés no era demostrar igualdad exacta, sino cuantificar la desviación introducida por la compresión con pérdida.

11.1.8 Criterio de Lectura de las Figuras

En todas las figuras, las líneas continuas representan resultados medidos hasta 25 qubits. A partir de ese punto, las líneas punteadas muestran proyecciones de tendencia hasta 28 qubits y las bandas sombreadas representan la incertidumbre asociada a esa estimación. Los fundamentos teóricos y los criterios estadísticos empleados para obtener estas predicciones se describen en el Apéndice B. Estas proyecciones se utilizan únicamente como apoyo interpretativo para discutir la dirección esperada de las curvas si se ampliara el tamaño del problema dentro del mismo régimen experimental; por tanto, no sustituyen la evidencia medida ni se emplean como base de las conclusiones principales.

11.1.9 Comparación Complementaria para Grover

Además de la comparación principal entre almacenamiento no comprimido, Blosc y ZFP, en Grover se incorporó una evaluación complementaria con la variante de parametrización reducida descrita en la Sección 10.6. Esta comparación adicional permitió distinguir entre el beneficio derivado de comprimir el vector de estado denso y el beneficio asociado a una reformulación

específica del algoritmo que reduce el volumen efectivo de datos manipulados. En esta subsección complementaria, la discusión se concentra en memoria y tiempo de ejecución, pues la exactitud ya queda determinada por la naturaleza exacta de la variante implementada y por la verificación de referencia efectuada en los experimentos de grover estándar.

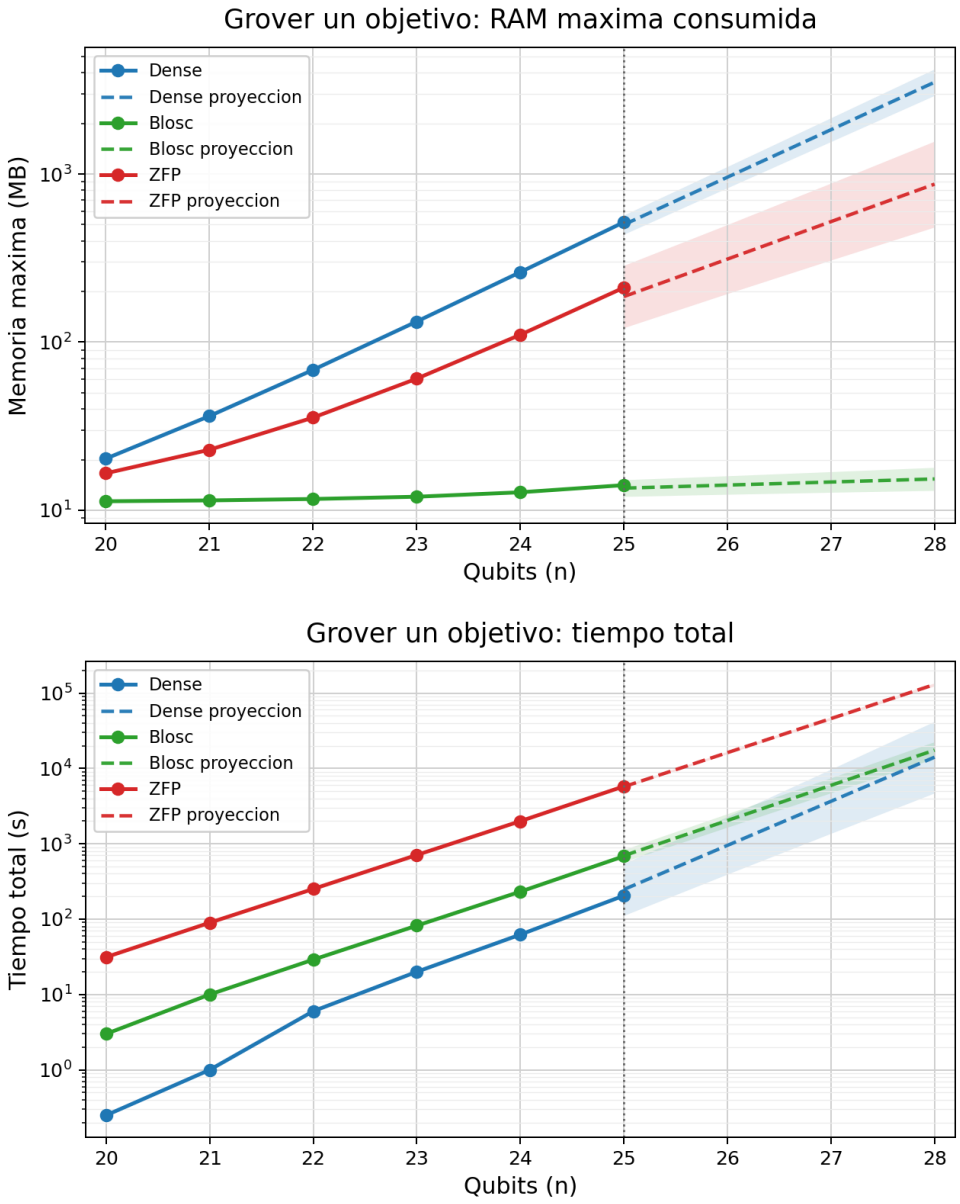
11.2 Resultados para Grover

Los experimentos con Grover representan el escenario más favorable para la estrategia sin pérdida propuesta. Como se explicó en la Sección 8.2, este algoritmo tiende a mantener una estructura de amplitudes altamente regular, lo que favorece la compresibilidad del vector de estado. A partir de esa base, esta sección analiza cómo dicha regularidad se refleja en las tres dimensiones evaluadas: consumo de memoria, tiempo de ejecución y exactitud numérica.

11.2.1 Objetivo Único

Figura 9

Grover con objetivo único: promedios sobre las posiciones $N/8$, $N/2$ y $7N/8$



La Figura 9 y la Tabla 8 muestran con claridad que Blosc reduce de forma importante el crecimiento de memoria observado en la referencia sin compresión. Mientras la referencia pasa de 20.3 MB a 516.7 MB entre 20 y 25 qubits, Blosc se mantiene en un intervalo mucho más estrecho, entre 11.3 MB y 14.1 MB. En términos de ahorro, esto equivale a una reducción que crece desde

44.2 % hasta 97.3 %.

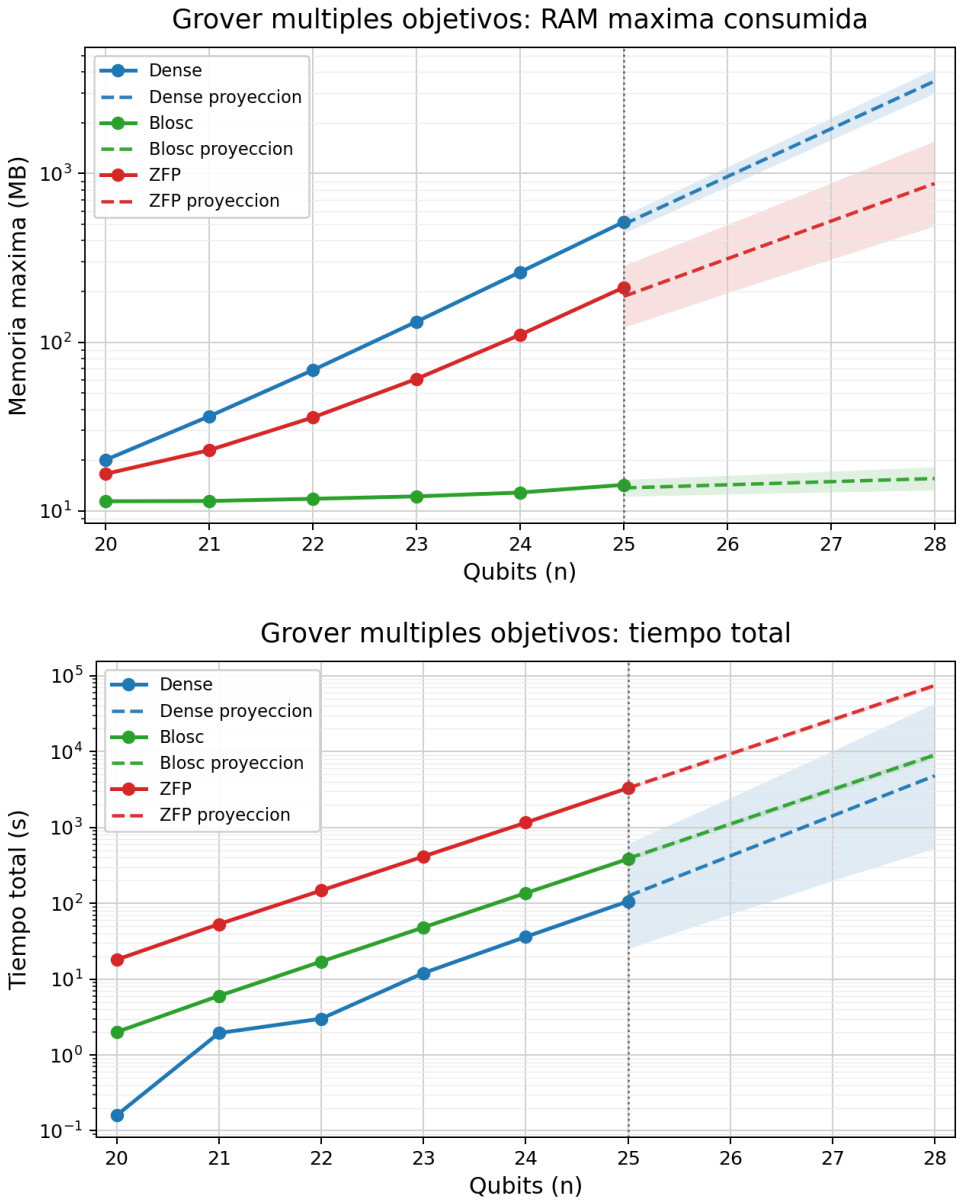
Este comportamiento indica que, en Grover, la compresión no depende solamente del tamaño del vector, sino también de su estructura interna. Dado que durante la ejecución aparecen pocas clases efectivas de amplitud, los bloques contienen patrones repetitivos o muy similares que el códec puede aprovechar. Como resultado, el costo adicional de metadatos y buffers auxiliares queda compensado por la reducción lograda en el almacenamiento.

En tiempo de ejecución, Blosc introduce una penalización respecto a la referencia en todos los tamaños evaluados. Esto era esperable, ya que la arquitectura comprimida debe descomprimir bloques, actualizarlos y volver a comprimirlos durante la simulación. Sin embargo, el crecimiento de esa sobrecarga no es proporcional al tamaño del problema. Por el contrario, el tiempo relativo disminuye de 12.00x en 20 qubits a 3.29x en 25 qubits. Esta tendencia sugiere que, a medida que aumenta la escala, el costo del códec se amortiza mejor frente al costo total de la simulación.

ZFP también reduce memoria, pero su comportamiento resulta menos favorable en este caso. En 25 qubits consume 210.8 MB, muy por encima de los 14.1 MB de Blosc, y además requiere 5788.27 s frente a 684.88 s. Esto muestra que, para un algoritmo como Grover, la compresión con pérdida no aporta una ventaja adicional significativa. La regularidad del estado ya es suficiente para que una estrategia sin pérdida consiga un ahorro muy alto sin introducir error numérico.

11.2.2 Multiobjetivo

Figura 10
Grover multiobjetivo con tres estados marcados



El caso multiobjetivo presenta el mismo comportamiento general, como se observa en la Figura 10 y en la Tabla 9. Aunque el oráculo cambia al marcar tres estados en lugar de uno, la estructura global del algoritmo sigue siendo lo suficientemente regular como para conservar una alta compresibilidad. Por esa razón, Blosc vuelve a mantener el consumo de memoria en un intervalo

muy reducido, entre 11.4 MB y 14.3 MB, mientras la referencia crece hasta 516.8 MB.

La comparación entre objetivo único y multiobjetivo permite ver que el ahorro de memoria no depende de una posición específica del estado marcado. En ambos casos, el comportamiento promedio es muy similar y los ahorros superan el 90 % a partir de 23 qubits. Además, Blosc mantiene error absoluto máximo nulo, lo que confirma que la reducción de memoria no altera el resultado de la simulación.

La diferencia más visible entre ambas variantes aparece en el tiempo de ejecución. En los tamaños mayores, el caso multiobjetivo resulta menos costoso que el de objetivo único tanto para la referencia como para las estrategias comprimidas. Dado que el patrón de memoria se mantiene prácticamente igual, esta diferencia parece estar asociada al trabajo concreto de la instancia evaluada y no a un cambio sustancial en la compresibilidad del estado.

ZFP repite aquí la misma tendencia observada en el caso anterior. Aunque reduce memoria frente a la referencia, queda por detrás de Blosc tanto en ahorro como en tiempo. En 25 qubits, por ejemplo, ZFP alcanza 59.1 % de ahorro y un tiempo relativo de 31.15x, mientras Blosc alcanza 97.2 % y 3.61x. En conjunto, estos resultados refuerzan que, para Grover, la estrategia sin pérdida es suficiente para capturar la estructura disponible de manera más eficiente.

11.2.3 Comparación Complementaria con la Variante Optimizada

Figura 11

Variante optimizada de Grover con objetivo único

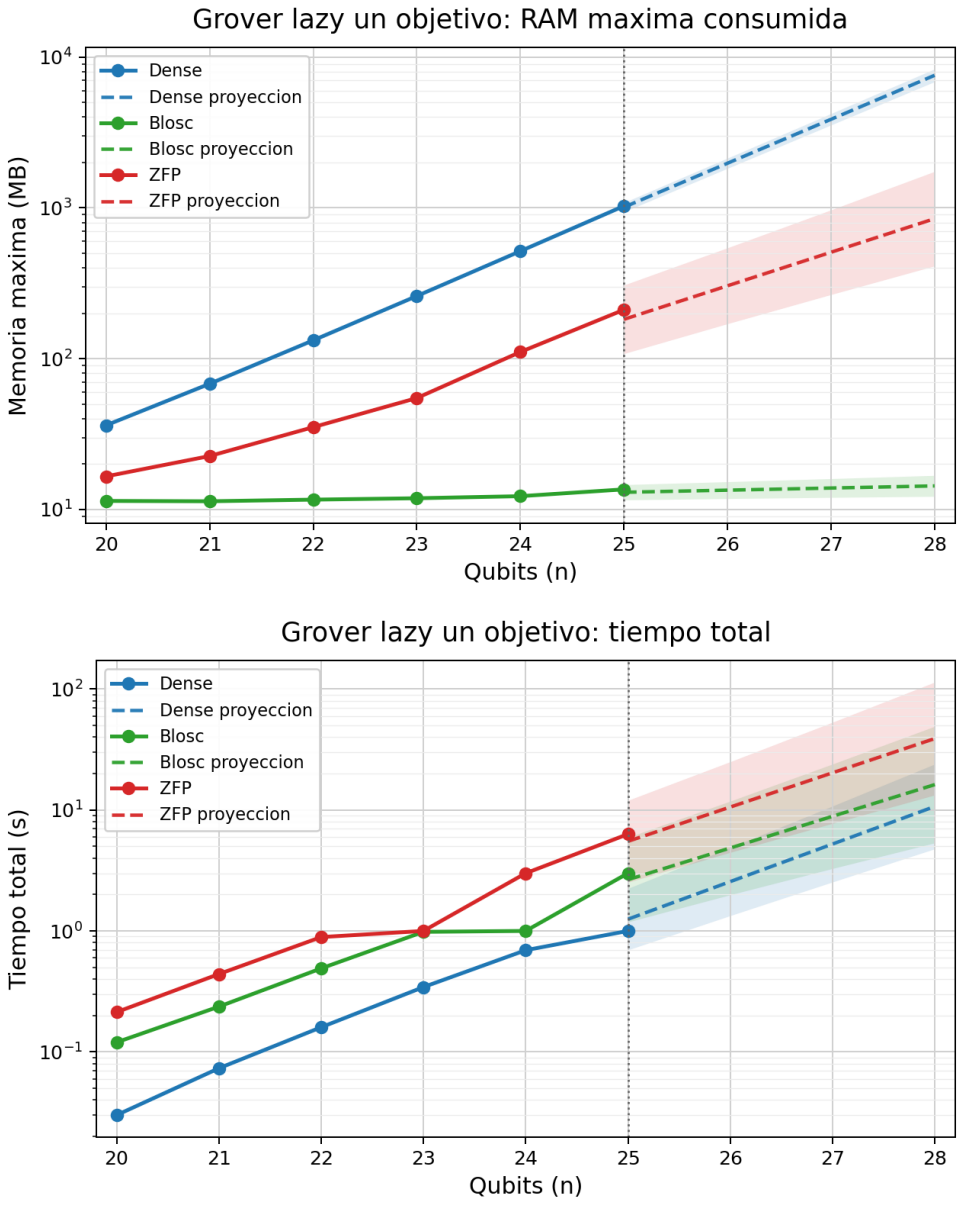
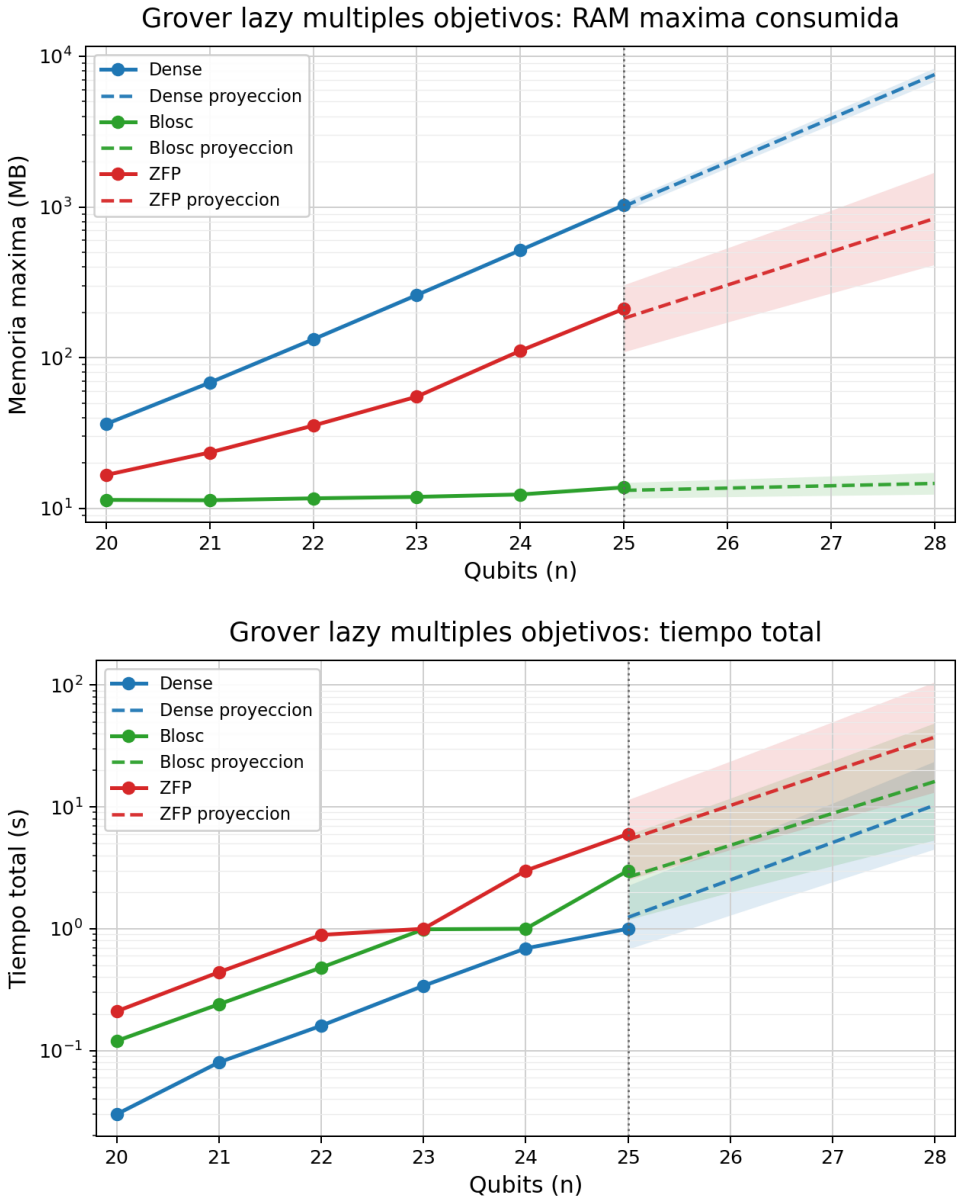


Figura 12

Variante optimizada de Grover multiobjetivo



Las Figuras 11 y 12, junto con las Tablas 10 y 11, permiten distinguir dos tipos de mejora que conviene no confundir. Por un lado, está la reducción de memoria obtenida al comprimir el vector de estado explícito. Por otro, está la optimización específica de Grover descrita en la Sección 10.6, que reformula la simulación de este algoritmo para reducir el trabajo necesario durante la ejecución.

La variante optimizada disminuye de forma drástica el tiempo de la referencia. En 25 qubits, la ejecución pasa de cientos de segundos en Grover estándar a 1.00 s en ambas configuraciones

optimizadas. Esto muestra que la mejora temporal más grande no proviene de la compresión, sino del aprovechamiento de una propiedad particular del algoritmo. Sin embargo, esa optimización no elimina por sí sola el crecimiento de memoria asociado a mantener una representación densa del estado. De hecho, la referencia optimizada todavía alcanza aproximadamente 1029 MB en 25 qubits.

En este contexto, Blosc conserva su aporte principal: reducir la huella de memoria sin alterar la exactitud. El consumo se mantiene entre 11.3 MB y 13.8 MB, con un ahorro de hasta 98.7 % en 25 qubits. La penalización temporal existe, pero en términos absolutos sigue siendo baja. La razón relativa, en cambio, fluctúa más porque la referencia optimizada es extremadamente rápida. Cuando el tiempo base se reduce tanto, cualquier costo fijo de compresión o de gestión de bloques tiene un peso mayor en la comparación relativa, aunque el tiempo total siga siendo pequeño.

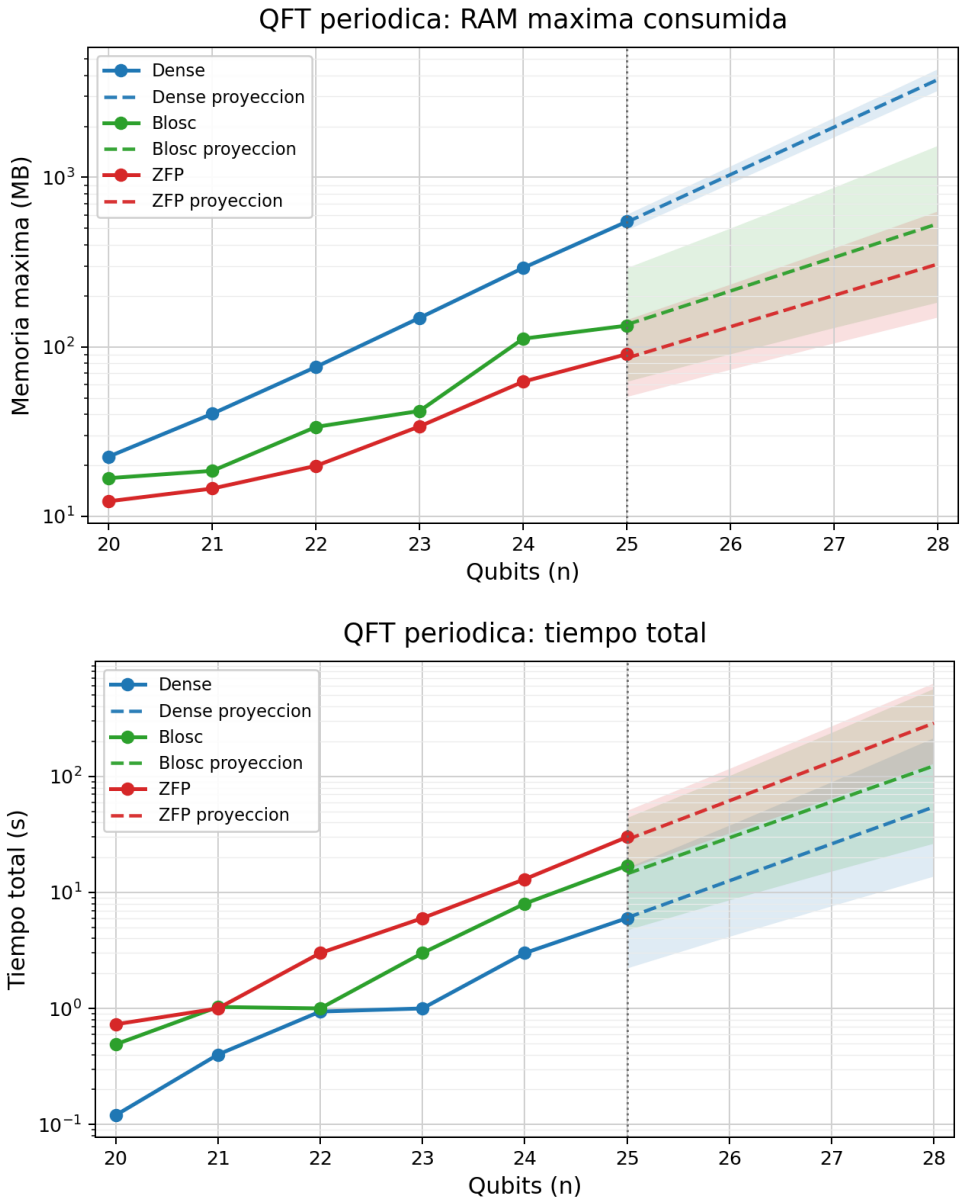
En consecuencia, esta comparación complementaria muestra que ambas estrategias resuelven problemas distintos. La compresión sin pérdida mejora la gestión de memoria del simulador, mientras que la variante reducida disminuye el trabajo específico de Grover. Su combinación resulta especialmente favorable para este algoritmo, pero no debe interpretarse como una propiedad general de cualquier circuito cuántico, sino como una ventaja asociada a una estructura particular de esta carga de trabajo.

11.3 Resultados para QFT

Los experimentos con QFT muestran un panorama menos favorable y más variable que el observado en Grover. Como se discutió en la Sección 8.3, la transformada cuántica de Fourier puede generar distribuciones de fase que reducen la redundancia exacta del vector de estado. Sin embargo, los resultados indican que el comportamiento final no depende solo del algoritmo, sino también de la estructura del estado de entrada. Por esta razón se evaluaron dos escenarios distintos: una superposición periódica y una superposición de alta entropía.

11.3.1 Superposición Periódica

Figura 13
QFT con superposición periódica



La Figura 13 y la Tabla 12 muestran que la superposición periódica produce un caso intermedio entre Grover y los escenarios menos compresibles. En este contexto, la compresión sí ofrece beneficios, pero de manera más moderada. Blosc alcanza ahorros de memoria entre 25.4 % y 75.7 %, manteniendo error nulo en todos los tamaños evaluados. ZFP, por su parte, obtiene

reducciones mayores en todos los casos y llega a 83.5 % de ahorro en 25 qubits.

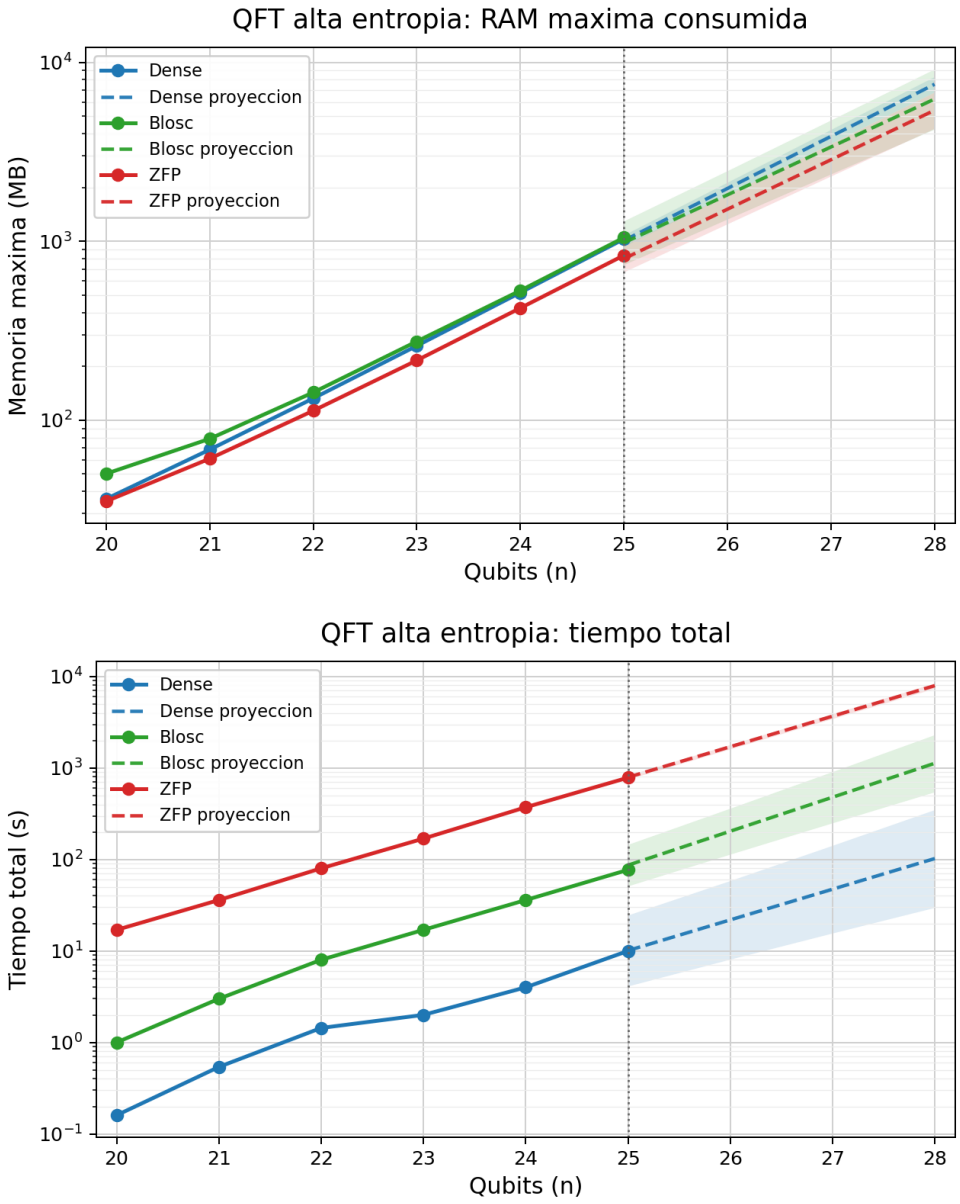
Estos resultados sugieren que la entrada periódica conserva cierta estructura aprovechable, aunque no con el mismo nivel de regularidad exacta presente en Grover. En otras palabras, el estado sigue siendo compresible, pero parte de esa compresibilidad parece provenir de correlaciones numéricas que una estrategia con pérdida puede explotar mejor que una estrategia estrictamente reversible. Por eso ZFP supera a Blosc en memoria en este escenario.

No obstante, la lectura cambia cuando se incorpora el tiempo de ejecución y la exactitud. En 25 qubits, ZFP reduce la memoria hasta 90.5 MB, mientras que Blosc la reduce a 133.4 MB. Sin embargo, ZFP tarda 30.01 s frente a 17.00 s de Blosc y, además, introduce un error del orden de 10^{-11}. Así, aunque ZFP ofrece el mayor ahorro espacial, Blosc conserva dos ventajas relevantes: coincidencia exacta con la referencia y menor sobrecarga temporal.

En consecuencia, este caso no muestra una estrategia claramente dominante en todos los criterios. ZFP resulta más atractivo cuando la prioridad principal es maximizar la reducción de memoria y se acepta una pequeña pérdida numérica. Blosc, en cambio, ofrece un equilibrio más conservador: ahorra menos memoria, pero preserva exactitud total y requiere menos tiempo de ejecución.

11.3.2 Superposición de Alta Entropía

Figura 14
QFT con superposición de alta entropía



La superposición de alta entropía cambia de forma clara el comportamiento de las estrategias evaluadas. Como se observa en la Figura 14 y en la Tabla 13, Blosc no consigue ahorro de memoria en ninguno de los tamaños analizados. Por el contrario, el consumo de la versión comprimida supera al de la referencia, aunque esa diferencia negativa se reduce a medida que aumenta el número de

qubits, pasando de -38.3 % a -1.8 % entre 20 y 25 qubits.

Este resultado es consistente con un estado denso y poco redundante. Cuando los bloques no contienen patrones exactos suficientes para compensar el costo de la compresión, los metadatos, buffers y estructuras auxiliares pasan a tener un peso visible en el balance final. En este caso, la arquitectura sigue funcionando correctamente, pero la naturaleza del estado hace que la compresión sin pérdida deje de ser ventajosa.

ZFP sí logra reducciones positivas de memoria, pero estas siguen siendo limitadas en relación con la sobrecarga temporal introducida. Los ahorros van de 3.0 % a 18.9 % entre 20 y 25 qubits, mientras que los tiempos relativos se mantienen en rangos muy altos. En 25 qubits, por ejemplo, la memoria baja de 1028.4 MB a 833.9 MB, pero el tiempo aumenta de 10.00 s a 783.28 s. Esto muestra que permitir pérdida numérica no garantiza por sí mismo una compresión útil, especialmente cuando la reducción obtenida es modesta frente al costo de procesamiento.

Este escenario permite delimitar con mayor precisión el alcance de la propuesta. La partición por bloques, la descompresión selectiva y la caché ayudan a administrar el acceso al vector, pero no crean redundancia donde no la hay. Por tanto, cuando el estado presenta alta entropía y poca estructura explotable, la compresión deja de ofrecer una ventaja práctica clara. Más que un problema de implementación, este resultado refleja un límite inherente a la relación entre la estructura del estado cuántico y la compresibilidad del vector.

11.4 Comparación Global de Estrategias

La lectura conjunta de las Figuras 9 hasta 14 y de las tablas completas del Apéndice C muestra que la utilidad de la compresión no depende únicamente del número de qubits ni del tamaño nominal del vector de estado. Lo que realmente determina su conveniencia es la estructura del estado cuántico y, en consecuencia, la cantidad de redundancia que puede ser aprovechada por la estrategia de almacenamiento.

En ese marco, Blosc ofrece el comportamiento más sólido cuando se exige exactitud numérica y el estado conserva regularidades suficientes para una compresión efectiva. ZFP, por

su parte, puede resultar competitivo cuando la prioridad principal es maximizar el ahorro de memoria y se acepta una pequeña pérdida de precisión. Sin embargo, los resultados también muestran que ese comportamiento favorable no se mantiene con la misma claridad en todas las cargas de trabajo consideradas. En los casos de mayor entropía y menor regularidad, ninguna de las estrategias comprimidas evaluadas ofrece un beneficio integral frente a la referencia. En consecuencia, la conveniencia del almacenamiento comprimido debe entenderse en función del régimen de compresibilidad asociado al circuito y no como una ventaja uniforme para cualquier patrón de simulación.

12 Conclusiones y Trabajo Futuro

12.1 Conclusiones

Los resultados obtenidos a lo largo de este trabajo muestran que la compresión sin pérdida puede incorporarse de manera viable a la gestión de memoria de un simulador cuántico tipo Schrödinger sin alterar la exactitud numérica de la simulación. La estrategia implementada sobre TMFQSfullstate, basada en almacenamiento por bloques, descompresión selectiva y reutilización temporal mediante caché, permitió reducir la huella de memoria manteniendo coincidencia exacta con la referencia no comprimida en todos los casos evaluados. En ese sentido, el estudio confirma que es posible intervenir la forma de almacenar el vector de estado sin modificar el modelo de simulación ni introducir aproximaciones sobre las amplitudes.

La evaluación experimental también permite precisar en qué condiciones esa integración resulta realmente favorable. En los circuitos de Grover, donde el estado conserva una estructura altamente regular, la compresión sin pérdida alcanzó sus mejores resultados y mostró una relación especialmente conveniente entre ahorro de memoria y costo temporal. En QFT, el comportamiento fue más dependiente de la naturaleza del estado de entrada: las superposiciones con patrones periódicos todavía ofrecieron oportunidades de compresión, mientras que los estados de alta entropía redujeron de forma marcada la efectividad del enfoque. Esta diferencia confirma que la utilidad práctica de la estrategia no está determinada solo por la escala del vector, sino por la forma en que la estructura del estado se traduce en redundancia aprovechable para el almacenamiento.

A partir de ello, la principal conclusión del trabajo no es que la compresión sin pérdida constituya una solución general para cualquier simulación cuántica de estado completo, sino que representa una alternativa técnicamente sólida cuando la memoria es la restricción dominante y la carga de trabajo conserva regularidades suficientes para ser explotadas. Bajo esas condiciones, la propuesta amplía la capacidad práctica de simulación exacta sin renunciar al formalismo de estado completo. Al mismo tiempo, los resultados dejan claro que ese beneficio no puede asumirse de

manera uniforme y que, en escenarios más irregulares o de alta entropía, no se observa una ventaja consistente frente a la ejecución convencional. Así, el valor de la estrategia queda delimitado con mayor precisión: no reside en prometer mejoras universales, sino en ofrecer una opción exacta y útil para un conjunto bien definido de regímenes de simulación.

12.2 Trabajo Futuro

Los resultados obtenidos abren varias líneas de trabajo orientadas a ampliar el alcance de la estrategia propuesta y a mejorar su desempeño en escenarios más exigentes. Una primera dirección consiste en extender la implementación hacia esquemas de paralelización que permitan distribuir de forma más eficiente tanto el almacenamiento como el procesamiento del vector de estado. En particular, el uso combinado de MPI y OpenMP permitiría explorar configuraciones de memoria distribuida y memoria compartida en las que la compresión por bloques no solo contribuya a reducir la huella de almacenamiento, sino también a hacer más viable la simulación en arquitecturas de mayor escala.

Otra línea de interés es la incorporación de mecanismos de análisis previo del circuito cuántico o de planificación de la ejecución. A partir de una lectura anticipada de la secuencia de compuertas, sería posible identificar patrones de acceso al vector de estado y reorganizar la ejecución para reducir la cantidad de accesos a los bloques comprimidos. En esa misma dirección, también podría estudiarse una disposición alternativa de las amplitudes dentro de los bloques, con el fin de agrupar componentes que tienden a ser accedidos conjuntamente y así disminuir la frecuencia de compresión y descompresión durante la simulación.

De manera complementaria, resulta prometedor estudiar estrategias con tamaños de bloque dinámicos. En lugar de mantener una partición fija para todo el vector de estado, podría definirse una organización adaptable que aproveche la relación entre tamaño del bloque, tasa de compresión y costo de acceso. Bajo este enfoque, las amplitudes o regiones del estado con accesos más frecuentes podrían almacenarse en bloques más pequeños, favoreciendo la rapidez de actualización, mientras que aquellas con menor actividad podrían mantenerse en bloques más grandes, orientados a mejorar

la compresión global. Esta posibilidad abre la puerta a políticas de almacenamiento más flexibles y sensibles al comportamiento real de la simulación.

Asimismo, sería valioso ampliar la comparación experimental tanto en estrategias de referencia como en tipos de circuitos evaluados. Por un lado, convendría contrastar la propuesta con otros enfoques de optimización de memoria, no solo basados en compresión, sino también en representaciones alternativas o esquemas híbridos. Por otro, sería pertinente incorporar algoritmos cuánticos adicionales y cargas de trabajo con estructuras más diversas, de modo que pueda caracterizarse con mayor precisión en qué clases de circuitos la compresión sin pérdida ofrece ventajas más consistentes.

Finalmente, una continuación natural de este trabajo consiste en profundizar en políticas adaptativas que permitan decidir, durante la ejecución, cuándo conviene comprimir, cómo particionar el estado y qué configuración utilizar según el comportamiento observado. Con ello, la compresión sin pérdida dejaría de operar como un esquema estático de almacenamiento y podría evolucionar hacia un componente dinámico de gestión de memoria, ajustado a las características de cada simulación.

Referencias Bibliográficas

Blosc Development Team. (2026). C-Blosc2 Documentation. Consultado el 29 de marzo de 2026, desde https://blosc.org/c-blosc2/c-blosc2.html

Boixo, S., Isakov, S. V., Smelyanskiy, V. N., & Neven, H. (2018). Simulation of low-depth quantum circuits as complex undirected graphical models [Preprint]. https://doi.org/10.48550/arXiv.1712.05384

Burtscher, M., & Ratanaworabhan, P. (2009). FPC: A High-Speed Compressor for Double-Precision Floating-Point Data. *IEEE Transactions on Computers*, *58*(1), 18-31. https://doi.org/10.1109/TC.2008.131

Chen, J., Zhang, F., Huang, C., Newman, M., & Shi, Y. (2018). Classical Simulation of Intermediate-Size Quantum Circuits [Preprint]. https://doi.org/10.48550/arXiv.1805.01450

Deutsch, D., & Jozsa, R. (1992). Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, *439*(1907), 553-558. https://doi.org/10.1098/rspa.1992.0167

Díaz, G., Saul, J., González Buendía, D., & Sarmiento, J. (2026). buendiagon/TMFQSfullstate: v1.1.0 [Source code archive of the TMFQSfullstate simulator]. https://doi.org/10.5281/zenodo.19663088

Díaz, G., Steffenel, L., Barrios, C., & Couturier, J. (2025). Memory Management Strategies for Software Quantum Simulators. *Quantum Reports*, *7*(3). https://doi.org/10.3390/quantum7030041

Díaz, G., Steffenel, L. A., Barrios, C. J., & Couturier, J.-F. (2024). How to Build a Software Quantum Simulator [Preprint; no peer-reviewed version confirmed as of 2026-04-01]. https://doi.org/10.20944/preprints202409.1497.v1

Díaz Toro, G. J. (2025, 3 de febrero). *Gestión de memoria en simuladores de computación cuántica de software* [Tesis doctoral]. Universidad Industrial de Santander. Consultado el 13 de mayo de 2025, desde https://noesis.uis.edu.co/items/357883df-3223-45b0-9848-f7681c5b0511

facebook/zstd contributors. (2026). facebook/zstd: Zstandard - Fast Real-Time Compression Algorithm. Consultado el 20 de enero de 2026, desde https://github.com/facebook/zstd

Gheorghiu, V. (2018). Quantum++: A modern C++ quantum computing library. *PLOS ONE*, *13*(12), e0208073. https://doi.org/10.1371/journal.pone.0208073

Google Quantum AI. (2025). qsimcirq Python Reference. Consultado el 27 de marzo de 2026, desde https://quantumai.google/reference/python/qsimcirq

Grover, L. K. (1996). A Fast Quantum Mechanical Algorithm for Database Search. *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, 212-219. https://doi.org/10.1145/237814.237866

Häner, T., & Steiger, D. S. (2017). 0.5 Petabyte Simulation of a 45-Qubit Quantum Circuit. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. https://doi.org/10.1145/3126908.3126947

Huffman, N., Pavlichin, D., & Weissman, T. (2024). Lossy Compression for Schrödinger-style Quantum Simulations [Preprint]. https://doi.org/10.48550/arXiv.2401.11088

Intel Corporation. (2026). Intel Quantum Simulator (Intel-QS) Documentation [Online documentation]. Consultado el 27 de marzo de 2026, desde https://intel-qs.readthedocs.io/en/docs/getting-started.html

Jamalidinan, S., & Cheshmi, K. (2025). Floating-Point Data Transformation for Lossless Compression [Preprint]. https://doi.org/10.48550/arXiv.2506.18062

Jaques, S., & Häner, T. (2022). Leveraging State Sparsity for More Efficient Quantum Simulations. *ACM Transactions on Quantum Computing*, *3*(3). https://doi.org/10.1145/3491248

Jones, T., Brown, A., Bush, I., & Benjamin, S. C. (2019). QuEST and High Performance Simulation of Quantum Computers. *Sci. Rep.*, *9*, 10736. https://doi.org/10.1038/s41598-019-47174-9

Lindstrom, P. (2014). Fixed-Rate Compressed Floating-Point Arrays. *IEEE Transactions on Visualization and Computer Graphics*, *20*(12), 2674-2683. https://doi.org/10.1109/TVCG.2014.2346458

Lindstrom, P., & Isenburg, M. (2006). Fast and Efficient Compression of Floating-Point Data. *IEEE Transactions on Visualization and Computer Graphics*, *12*(5), 1245-1250. https://doi.org/10.1109/TVCG.2006.143

lz4 contributors. (2026). lz4/lz4: Extremely Fast Compression Algorithm. Consultado el 20 de enero de 2026, desde https://github.com/lz4/lz4

Markov, I. L., & Shi, Y. (2008). Simulating Quantum Computation by Contracting Tensor Networks. *SIAM Journal on Computing*, *38*(3), 963-981. https://doi.org/10.1137/050644756

Masui, K., Amiri, M., Connor, L., Deng, M., Fandino, M., Höfer, C., Halpern, M., Hanna, D., Hincks, A. D., Hinshaw, G., Parra, J. M., Newburgh, L. B., Shaw, J. R., & Vanderlinde, K. (2015). A compression scheme for radio data in high performance computing. *Astronomy and Computing*, *12*, 181-190. https://doi.org/10.1016/j.ascom.2015.07.002

Miller, D. M., & Thornton, M. A. (2006). QMDD: A Decision Diagram Structure for Reversible and Quantum Circuits. *36th International Symposium on Multiple-Valued Logic (ISMVL'06)*, 30-30. https://doi.org/10.1109/ISMVL.2006.35

Nielsen, M. A., & Chuang, I. L. (2010). *Quantum Computation and Quantum Information* (10th Anniversary Edition). Cambridge University Press.

Pednault, E., Gunnels, J. A., Nannicini, G., Horesh, L., Magerlein, T., Solomonik, E., Draeger, E. W., Holland, E. T., & Wisnieff, R. (2017). Pareto-Efficient Quantum Circuit Simulation Using Tensor Contraction Deferral. https://arxiv.org/abs/1710.05867

Qiskit Development Team. (2025). Qiskit Aer Documentation. Consultado el 27 de marzo de 2026, desde https://qiskit.github.io/qiskit-aer/

Quantumlib contributors. (2025). qsim: Fast C++ and Python library for state-vector simulation of quantum circuits. Consultado el 27 de marzo de 2026, desde https://github.com/quantumlib/qsim

QuEST-Kit contributors. (2026). QuEST: Quantum Exact Simulation Toolkit. Consultado el 27 de marzo de 2026, desde https://github.com/QuEST-Kit/QuEST

Simon, D. R. (1997). On the Power of Quantum Computation. *SIAM Journal on Computing*, *26*(5), 1474-1483. https://doi.org/10.1137/S0097539796298637

Szabłowski, P. J. (2021). Understanding mathematics of Grover's algorithm. *Quantum Information Processing*, *20*(5). https://doi.org/10.1007/s11128-021-03125-w

Viamontes, G. F., Markov, I. L., & Hayes, J. P. (2003). Improving Gate-Level Simulation of Quantum Circuits. *Quantum Information Processing*, *2*(5), 347-380. https://doi.org/10.1023/B:QINP.0000022725.70000.4a

Vidal, G. (2003). Efficient Classical Simulation of Slightly Entangled Quantum Computations. *Phys. Rev. Lett.*, *91*(14), 147902. https://doi.org/10.1103/PhysRevLett.91.147902

Vinkhuijzen, L., Coopmans, T., Elkouss, D., Dunjko, V., & Laarman, A. (2023). LIMDD: A Decision Diagram for Simulation of Quantum Computing Including Stabilizer States. *Quantum*, *7*, 1108. https://doi.org/10.22331/q-2023-09-11-1108

Wu, X.-C., Di, S., Cappello, F., Finkel, H., Alexeev, Y., & Chong, F. T. (2018). Amplitude-Aware Lossy Compression for Quantum Circuit Simulation. *Proceedings of the 4th International Workshop on Data Reduction for Big Scientific Data (DRBSD-4) at SC18*. https://sc18.supercomputing.org/proceedings/workshops/workshop_files/ws_drbsd112s1-file1.pdf

Wu, X.-C., Di, S., Dasgupta, E. M., Cappello, F., Finkel, H., Alexeev, Y., & Chong, F. T. (2019). Full-State Quantum Circuit Simulation by Using Data Compression. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'19)*. https://doi.org/10.1145/3295500.3356155

Zhang, B., Fang, B., Ye, F., Guo, L., Tallent, N., & Tao, D. (2025). BMQSim: Overcoming Memory Constraints in Quantum Circuit Simulation with a High-Fidelity Compression Framework. *Proceedings of the 39th ACM International Conference on Supercomputing*. https://doi.org/10.1145/3721145.3725747

zlib authors. (2022). zlib Technical Details. Consultado el 20 de enero de 2026, desde https://zlib.net/zlib_tech.html

Zstandard project. (2026). Zstandard - Real-Time Data Compression Algorithm. Consultado el 20 de enero de 2026, desde https://facebook.github.io/zstd/

Apéndices

A Derivación de la Parametrización Reducida de Grover

La idea central es que, bajo las hipótesis habituales del algoritmo, la evolución no necesita seguir de manera independiente las N amplitudes del vector de estado. Debido a la simetría entre estados marcados y no marcados, basta con describir la dinámica mediante dos parámetros: una amplitud común para los estados marcados y otra para los no marcados.

Sea $W \subseteq \{0, \ldots, N-1\}$ el conjunto de estados marcados por el oráculo, con $M = |W|$. En la versión estándar del algoritmo, el estado inicial es la superposición uniforme

$$|s\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle \,,$$

y el oráculo actúa cambiando el signo de las amplitudes asociadas a los estados de W.

Bajo estas condiciones, todos los estados marcados tienen inicialmente la misma amplitud, y lo mismo ocurre con todos los estados no marcados. Además, tanto el oráculo como el operador de difusión preservan esa simetría: el primero afecta por igual a todos los estados marcados, y el segundo transforma cada amplitud únicamente en función del promedio global. Por ello, durante toda la evolución ideal del algoritmo, las amplitudes pueden agruparse en solo dos clases:

$$a_x^{(t)} = \begin{cases} u_t, & x \in W, \\ v_t, & x \notin W, \end{cases}$$

donde u_t representa la amplitud común de los estados marcados en la iteración t, y v_t la amplitud común de los no marcados.

Esta misma idea puede expresarse en términos de una base reducida del subespacio relevante.

Definiendo los vectores normalizados

$$|w\rangle = \frac{1}{\sqrt{M}} \sum_{x \in W} |x\rangle , \qquad |r\rangle = \frac{1}{\sqrt{N-M}} \sum_{x \notin W} |x\rangle ,$$

el estado inicial uniforme se descompone como

$$|s\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle = \sqrt{\frac{M}{N}}\ |w\rangle + \sqrt{\frac{N-M}{N}}\ |r\rangle .$$

En consecuencia, la evolución ideal de Grover queda confinada al subespacio generado por $\{|w\rangle , |r\rangle\}$, y cualquier estado del algoritmo puede escribirse como

$$|\psi_t\rangle = \alpha_t |w\rangle + \beta_t |r\rangle .$$

La relación entre ambas representaciones es directa. Como $|w\rangle$ distribuye uniformemente la amplitud sobre los M estados marcados y $|r\rangle$ sobre los $N - M$ restantes, se tiene

$$u_t = \frac{\alpha_t}{\sqrt{M}}, \qquad v_t = \frac{\beta_t}{\sqrt{N-M}}.$$

Por tanto, seguir la evolución en términos de (α_t, β_t) o en términos de (u_t, v_t) es equivalente.

Acción del Oráculo

El oráculo de Grover cambia el signo de las amplitudes correspondientes a los estados marcados y deja intactas las demás. Por tanto, si antes del oráculo el estado está descrito por (u_t, v_t), después del oráculo las amplitudes quedan como

$$u'_t = -u_t, \qquad v'_t = v_t.$$

Acción del Operador de Difusión

El operador de difusión viene dado por

$$D = 2\,|s\rangle\langle s| - I,$$

y puede interpretarse como una inversión de cada amplitud respecto al promedio global. Si μ_t denota el promedio de amplitud después de aplicar el oráculo, entonces la difusión transforma cada componente según la regla

$$x \mapsto 2\mu_t - x.$$

Después del oráculo, el promedio de amplitud es

$$\mu_t = \frac{1}{N}\left(\sum_{x\in W}(-u_t) + \sum_{x\notin W} v_t\right) = \frac{-Mu_t + (N-M)v_t}{N}.$$

Aplicando ahora la difusión:

Para un estado marcado, cuya amplitud después del oráculo es $-u_t$,

$$u_{t+1} = 2\mu_t - (-u_t) = 2\mu_t + u_t.$$

Sustituyendo μ_t,

$$u_{t+1} = 2\left(\frac{-Mu_t + (N-M)v_t}{N}\right) + u_t = \frac{N-2M}{N}u_t + \frac{2(N-M)}{N}v_t.$$

Para un estado no marcado, cuya amplitud después del oráculo es v_t,

$$v_{t+1} = 2\mu_t - v_t.$$

Sustituyendo nuevamente μ_t,

$$v_{t+1} = 2\left(\frac{-Mu_t + (N-M)v_t}{N}\right) - v_t = -\frac{2M}{N}u_t + \frac{N-2M}{N}v_t.$$

En consecuencia, una iteración completa de Grover queda descrita por la recurrencia

$$\begin{bmatrix} u_{t+1} \\ v_{t+1} \end{bmatrix} = \begin{bmatrix} \frac{N-2M}{N} & \frac{2(N-M)}{N} \\ -\frac{2M}{N} & \frac{N-2M}{N} \end{bmatrix} \begin{bmatrix} u_t \\ v_t \end{bmatrix}.$$

Condiciones Iniciales y Reconstrucción del Estado

Como el algoritmo parte de la superposición uniforme, en $t = 0$ todas las amplitudes son iguales a $1/\sqrt{N}$. Por tanto,

$$u_0 = v_0 = \frac{1}{\sqrt{N}}.$$

A partir de la recurrencia anterior, cada iteración puede resolverse actualizando únicamente los parámetros u_t y v_t, sin necesidad de aplicar el oráculo y la difusión sobre las N amplitudes del vector de estado. Si en algún momento se requiere reconstruir el estado completo, basta asignar

$$a_x^{(t)} = \begin{cases} u_t, & x \in W, \\ v_t, & x \notin W. \end{cases}$$

Alcance de la Reducción

La parametrización reducida depende de una propiedad muy específica de Grover: la evolución preserva la igualdad de amplitud dentro de las clases “marcada” y “no marcada”. Esa propiedad se cumple cuando el algoritmo parte de la superposición uniforme y el oráculo solo introduce un cambio de fase sobre los estados marcados. Bajo estas hipótesis, la dinámica efectiva

queda contenida en un subespacio de dimensión dos.

B Metodología de Proyección en las Gráficas Experimentales

Las gráficas de la evaluación experimental distinguen, para cada caso analizado, dos magnitudes: el tiempo total de ejecución y la memoria máxima consumida. En todas ellas, el eje x representa el número de qubits n, mientras que el eje y se presenta en escala logarítmica. Esta elección responde a que tanto el tamaño del vector de estado como el costo de simularlo tienden a crecer de manera multiplicativa a medida que aumenta n. Usar una escala logarítmica permite, por tanto, comparar en una misma figura valores que difieren por varios órdenes de magnitud sin perder legibilidad.

En cada curva, las líneas continuas corresponden a mediciones obtenidas directamente en los experimentos realizados. A partir del último punto medido, las líneas discontinuas muestran una proyección del comportamiento esperado para tamaños no evaluados de forma directa. La banda sombreada indica el intervalo de predicción del 95 %, y la línea vertical punteada marca precisamente el límite entre la región observada y la región proyectada. De este modo, la figura permite distinguir con claridad qué parte de la curva proviene de datos medidos y cuál debe interpretarse como una estimación.

Modelo de Proyección

La proyección se construyó de manera independiente para cada combinación de experimento, estrategia de almacenamiento y métrica. Si y_i denota el tiempo o la memoria medidos para un número de qubits n_i, el primer paso consiste en transformar la variable de respuesta al espacio logarítmico:

$$z_i = \log(y_i).$$

Esta transformación se utiliza porque el crecimiento de tiempo y memoria en simulación cuántica de estado completo puede aproximarse razonablemente mediante una tendencia exponencial

respecto al número de qubits. Al pasar al espacio logarítmico, esa relación se vuelve aproximadamente lineal, lo que permite ajustarla con un modelo más simple e interpretable. Además, al regresar después a la escala original mediante la función $\exp(\cdot)$, las predicciones conservan valores positivos y mantienen una interpretación coherente con magnitudes como tiempo y memoria.

En los casos en que una misma figura resume varias variantes de un experimento, como ocurre con Grover para los objetivos ubicados en $N/8$, $N/2$ y $7N/8$, primero se obtiene un valor representativo por qubit y estrategia mediante la media geométrica. Esta elección es consistente con el ajuste en espacio logarítmico, ya que equivale a promediar los valores transformados y luego volver a la escala original.

Sobre los valores así obtenidos se ajusta una regresión lineal de la forma

$$z_i = \beta_0 + \beta_1 n_i + \varepsilon_i,$$

donde ε_i recoge la variación que no queda explicada por la tendencia aproximadamente exponencial del fenómeno.

La estimación de los parámetros se realiza por mínimos cuadrados ordinarios. Si X es la matriz de diseño, formada por una columna de unos y una columna con los valores de n_i, entonces

$$\hat{\beta} = (X^\top X)^{-1} X^\top z.$$

Para un nuevo tamaño n_0, con $x_0 = [1, n_0]^\top$, la predicción en espacio logarítmico viene dada por

$$\hat{z}_0 = x_0^\top \hat{\beta}.$$

Finalmente, el valor proyectado que se muestra en la gráfica se obtiene al volver a la escala original:

$$\hat{y}_0 = \exp(\hat{z}_0).$$

Estimación de la Incertidumbre

La incertidumbre asociada a la proyección también se calcula en espacio logarítmico. Una vez ajustado el modelo, la varianza residual se estima como

$$s^2 = \frac{1}{m-p} \sum_{i=1}^{m} (z_i - \hat{z}_i)^2,$$

donde m es el número de observaciones utilizadas en el ajuste y $p = 2$ corresponde al número de parámetros del modelo. En las curvas presentadas en este trabajo se emplean seis mediciones por estrategia, correspondientes a $q20$ hasta $q25$, por lo que los grados de libertad son $m - p = 4$.

Para un nuevo valor n_0, el error estándar de predicción se calcula como

$$s_{\text{pred}}(n_0) = s\sqrt{1 + x_0^\top (X^\top X)^{-1} x_0}.$$

Esta expresión incorpora dos fuentes de incertidumbre. Por un lado, el término

$$x_0^\top (X^\top X)^{-1} x_0$$

refleja que la proyección se vuelve menos precisa a medida que el punto estimado se aleja de la región en la que se cuenta con mediciones. Por otro, el término adicional 1 indica que no se está construyendo solo un intervalo de confianza para la media de la tendencia, sino un intervalo de predicción para una posible observación futura.

Con ello, el intervalo de predicción del 95 % en espacio logarítmico se expresa como

$$\hat{z}_0 \pm t_{0{,}975,\, m-p}\; s_{\text{pred}}(n_0),$$

donde $t_{0{,}975,\, m-p}$ es el valor crítico de la distribución t de Student con $m - p$ grados de libertad.

Para representar este intervalo en la escala original, se aplica la función exponencial a sus

dos extremos:

$$\left[\exp\left(\hat{z}_0 - t_{0{,}975,\,m-p}\, s_{\mathrm{pred}}(n_0)\right),\, \exp\left(\hat{z}_0 + t_{0{,}975,\,m-p}\, s_{\mathrm{pred}}(n_0)\right)\right].$$

Se emplea la distribución t de Student, y no una aproximación normal, porque cada curva se ajusta con un número reducido de observaciones. Esta elección permite incorporar de forma más adecuada la incertidumbre asociada a la estimación de la varianza residual. Del mismo modo, se utiliza un intervalo de predicción y no únicamente un intervalo de confianza, ya que la banda sombreada busca indicar un rango plausible para futuras ejecuciones y no solo la incertidumbre de la tendencia media ajustada.

C Tablas Completas de la Evaluación Experimental

Tabla 8
Grover con objetivo único: promedios sobre las posiciones $N/8$, $N/2$ y $7N/8$

Qubits	Estrategia	Mem. pico (MB)	Ahorro mem. (%)	Tiempo (s)	Tiempo rel. (x)	Error
20	Referencia	20.3	0.0	0.25	1.00	0
20	Blosc	11.3	44.2	3.00	12.00	0
20	ZFP	16.6	18.2	31.34	125.38	$1{,}20 \times 10^{-9}$
21	Referencia	36.3	0.0	1.00	1.00	0
21	Blosc	11.4	68.5	10.00	10.00	0
21	ZFP	22.9	37.0	89.70	89.70	$1{,}04 \times 10^{-9}$
22	Referencia	68.4	0.0	6.00	1.00	0
22	Blosc	11.7	82.9	29.01	4.83	0
22	ZFP	35.5	48.0	251.42	41.88	$1{,}23 \times 10^{-9}$
23	Referencia	132.5	0.0	20.01	1.00	0
23	Blosc	12.0	90.9	82.02	4.10	0
23	ZFP	60.6	54.3	710.25	35.49	$1{,}85 \times 10^{-9}$
24	Referencia	260.6	0.0	62.37	1.00	0
24	Blosc	12.8	95.1	231.72	3.72	0
24	ZFP	110.5	57.6	1996.71	32.02	$2{,}58 \times 10^{-9}$
25	Referencia	516.7	0.0	208.12	1.00	0
25	Blosc	14.1	97.3	684.88	3.29	0
25	ZFP	210.8	59.2	5788.27	27.81	$3{,}21 \times 10^{-9}$

Tabla 9
Grover multiobjetivo con tres estados marcados

Qubits	Estrategia	Mem. pico (MB)	Ahorro mem. (%)	Tiempo (s)	Tiempo rel. (x)	Error
20	Referencia	20.1	0.0	0.16	1.00	0
20	Blosc	11.4	43.2	2.00	12.51	0
20	ZFP	16.6	17.5	18.01	112.54	$9{,}09 \times 10^{-10}$
21	Referencia	36.4	0.0	1.94	1.00	0
21	Blosc	11.5	68.6	6.00	3.09	0
21	ZFP	22.9	37.1	53.02	27.33	$3{,}94 \times 10^{-10}$
22	Referencia	68.5	0.0	3.00	1.00	0
22	Blosc	11.8	82.8	17.00	5.67	0
22	ZFP	35.9	47.6	147.05	49.00	$5{,}43 \times 10^{-10}$
23	Referencia	132.6	0.0	12.01	1.00	0
23	Blosc	12.2	90.8	48.01	4.00	0
23	ZFP	60.6	54.3	414.15	34.49	$8{,}28 \times 10^{-10}$
24	Referencia	260.7	0.0	36.02	1.00	0
24	Blosc	12.9	95.1	136.03	3.78	0
24	ZFP	110.8	57.5	1155.41	32.08	$9{,}91 \times 10^{-10}$
25	Referencia	516.8	0.0	106.06	1.00	0
25	Blosc	14.3	97.2	383.11	3.61	0
25	ZFP	211.3	59.1	3304.05	31.15	$1{,}69 \times 10^{-9}$

Tabla 10
Variante optimizada de Grover con objetivo único

Qubits	Estrategia	Mem. pico (MB)	Ahorro mem. (%)	Tiempo (s)	Tiempo rel. (x)	Error
20	Referencia	36.2	0.0	0.03	1.00	0
20	Blosc	11.4	68.5	0.12	4.00	0
20	ZFP	16.6	54.2	0.21	7.11	$2{,}18 \times 10^{-11}$
21	Referencia	68.2	0.0	0.07	1.00	0
21	Blosc	11.3	83.4	0.24	3.23	0
21	ZFP	22.6	66.8	0.44	6.00	$1{,}06 \times 10^{-11}$
22	Referencia	132.3	0.0	0.16	1.00	0
22	Blosc	11.6	91.2	0.49	3.06	0
22	ZFP	35.1	73.4	0.89	5.56	$3{,}04 \times 10^{-11}$
23	Referencia	260.2	0.0	0.34	1.00	0
23	Blosc	11.9	95.4	0.98	2.86	0
23	ZFP	54.8	78.9	1.00	2.91	$1{,}71 \times 10^{-11}$
24	Referencia	516.4	0.0	0.69	1.00	0
24	Blosc	12.2	97.6	1.00	1.44	0
24	ZFP	110.7	78.6	3.00	4.33	$3{,}35 \times 10^{-11}$
25	Referencia	1028.8	0.0	1.00	1.00	0
25	Blosc	13.6	98.7	3.00	3.00	0
25	ZFP	211.0	79.5	6.34	6.34	$7{,}51 \times 10^{-12}$

Tabla 11
Variante optimizada de Grover multiobjetivo

Qubits	Estrategia	Mem. pico (MB)	Ahorro mem. (%)	Tiempo (s)	Tiempo rel. (x)	Error
20	Referencia	36.2	0.0	0.03	1.00	0
20	Blosc	11.4	68.6	0.12	4.00	0
20	ZFP	16.6	54.1	0.21	7.00	$1{,}68 \times 10^{-11}$
21	Referencia	68.2	0.0	0.08	1.00	0
21	Blosc	11.3	83.4	0.24	3.00	0
21	ZFP	23.5	65.6	0.44	5.50	$1{,}07 \times 10^{-11}$
22	Referencia	132.3	0.0	0.16	1.00	0
22	Blosc	11.6	91.2	0.48	3.00	0
22	ZFP	35.4	73.2	0.89	5.56	$1{,}73 \times 10^{-11}$
23	Referencia	260.2	0.0	0.34	1.00	0
23	Blosc	11.9	95.4	0.99	2.91	0
23	ZFP	55.1	78.8	1.00	2.94	$1{,}29 \times 10^{-11}$
24	Referencia	516.6	0.0	0.69	1.00	0
24	Blosc	12.4	97.6	1.00	1.45	0
24	ZFP	111.1	78.5	3.00	4.35	$1{,}65 \times 10^{-11}$
25	Referencia	1028.9	0.0	1.00	1.00	0
25	Blosc	13.8	98.7	3.00	3.00	0
25	ZFP	210.9	79.5	6.00	6.00	$1{,}33 \times 10^{-11}$

Tabla 12
QFT con superposición periódica

Qubits	Estrategia	Mem. pico (MB)	Ahorro mem. (%)	Tiempo (s)	Tiempo rel. (x)	Error
20	Referencia	22.5	0.0	0.12	1.00	0
20	Blosc	16.8	25.4	0.49	4.08	0
20	ZFP	12.3	45.6	0.73	6.08	$3{,}79 \times 10^{-11}$
21	Referencia	40.3	0.0	0.40	1.00	0
21	Blosc	18.5	54.0	1.03	2.57	0
21	ZFP	14.6	63.8	1.00	2.50	$3{,}79 \times 10^{-11}$
22	Referencia	76.1	0.0	0.94	1.00	0
22	Blosc	33.7	55.7	1.00	1.06	0
22	ZFP	19.8	73.9	3.00	3.19	$3{,}79 \times 10^{-11}$
23	Referencia	148.3	0.0	1.00	1.00	0
23	Blosc	41.9	71.8	3.00	3.00	0
23	ZFP	33.9	77.1	6.00	6.00	$3{,}79 \times 10^{-11}$
24	Referencia	292.3	0.0	3.00	1.00	0
24	Blosc	111.7	61.8	8.00	2.67	0
24	ZFP	62.3	78.7	13.00	4.33	$3{,}79 \times 10^{-11}$
25	Referencia	548.4	0.0	6.00	1.00	0
25	Blosc	133.4	75.7	17.00	2.83	0
25	ZFP	90.5	83.5	30.01	5.00	$3{,}79 \times 10^{-11}$

Tabla 13
QFT con superposición de alta entropía

Qubits	Estrategia	Mem. pico (MB)	Ahorro mem. (%)	Tiempo (s)	Tiempo rel. (x)	Error
20	Referencia	36.3	0.0	0.16	1.00	0
20	Blosc	50.2	-38.3	1.00	6.25	0
20	ZFP	35.2	3.0	17.00	106.28	$3{,}29 \times 10^{-11}$
21	Referencia	68.5	0.0	0.54	1.00	0
21	Blosc	78.7	-15.0	3.00	5.56	0
21	ZFP	60.9	11.0	36.01	66.69	$2{,}64 \times 10^{-11}$
22	Referencia	132.4	0.0	1.44	1.00	0
22	Blosc	142.9	-7.9	8.00	5.56	0
22	ZFP	112.9	14.7	80.03	55.57	$3{,}53 \times 10^{-11}$
23	Referencia	260.3	0.0	2.00	1.00	0
23	Blosc	275.5	-5.8	17.01	8.50	0
23	ZFP	215.9	17.1	169.06	84.49	$2{,}82 \times 10^{-11}$
24	Referencia	516.3	0.0	4.00	1.00	0
24	Blosc	529.8	-2.6	36.01	9.00	0
24	ZFP	423.4	18.0	372.12	92.98	$3{,}79 \times 10^{-11}$
25	Referencia	1028.4	0.0	10.00	1.00	0
25	Blosc	1047.2	-1.8	77.03	7.70	0
25	ZFP	833.9	18.9	783.28	78.30	$3{,}01 \times 10^{-11}$