RESULTADOS ALCATRAZ

OPORTUNIDADES DEL NUEVO MODELO PRODUCTIVO ASTURIANO HABILITADO POR LA INVESTIGACIÓN EN COMPUTACIÓN Y ARQUITECTURAS AVANZADAS

En	Gijón	, a	29	de	Diciembre	de	2023

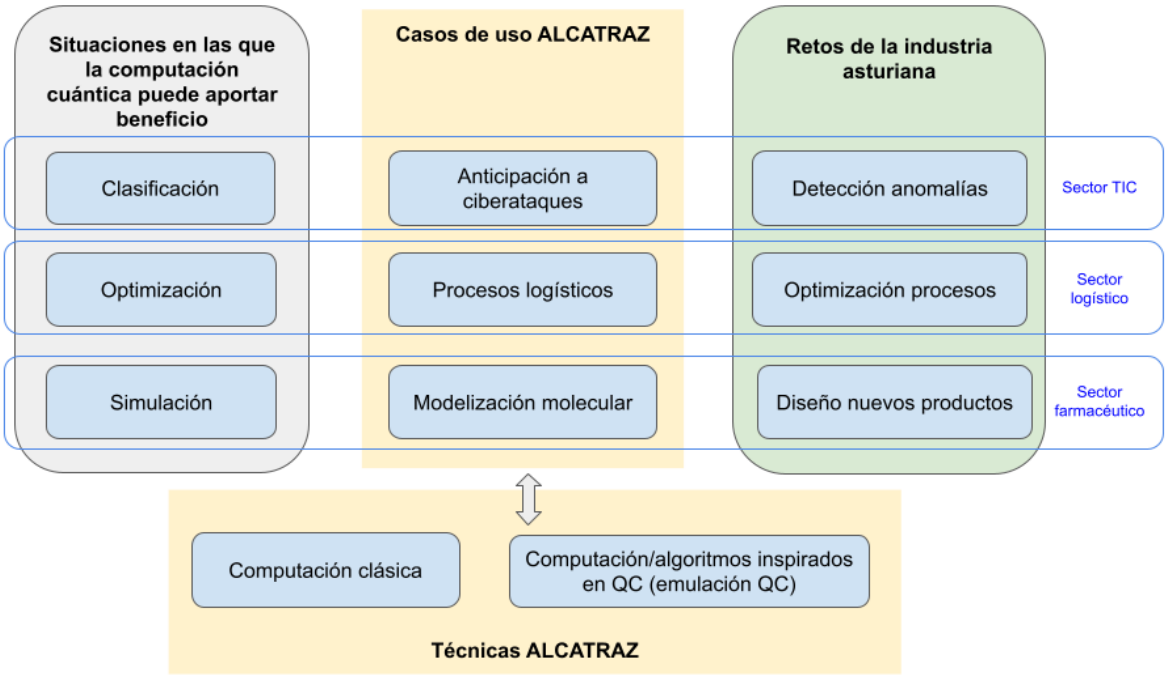
1. MEMORIA TÉCNICA

Introducción

El proyecto ALCATRAZ (oportunidAdes del nuevo modeLo produCtivo Asturiano habilitado por la investigación en compuTación y arquitectuRas AvanZadas) es ejecutado por Fundación CTIC – Centro Tecnológico entre el 1 de enero de 2021 y el 31 de diciembre de 2023. **Esta memoria recoge la descripción de los trabajos realizados durante todo el proyecto**, por tanto, incluye las tres anualidades: 2021, 2022 y 2023.

El proyecto ALCATRAZ tiene como objetivo general la investigación en la resolución de problemas prácticos de una alta complejidad, que estén estrechamente ligados al tejido industrial regional asociados al propio modelo productivo a través del estudio de cómo de viable y eficiente sea la aplicación de distintas técnicas de computación cuántica en contraposición de técnicas clásicas.

En la siguiente figura se presenta de forma visual un resumen de dicho planteamiento.



Justificación 2021

A continuación, se describen las tareas realizadas en este hito de trabajo y los resultados conseguidos.

PT1: Análisis de los retos a resolver

Este paquete de trabajo se organiza en torno a dos tareas, las cuales han sido iniciadas y finalizadas en su totalidad en esta anualidad:

- T1.1: Identificación de los retos
- T1.2: Análisis de los subproblemas

A continuación, se describen las acciones desarrolladas en cada una de las tareas y los principales resultados alcanzados en cada una de ellas.

T1.1: Identificación de los retos

Basándose en tres problemas matemáticos susceptibles de ser analizados desde el punto de vista de la computación cuántica con aplicación directa en la industria (clasificación, optimización y simulación) y teniendo en cuenta el contexto socio-económico en el que se encuentra Asturias, se ha procedido a identificar los retos industriales de mayor interés sobre los que validar el objetivo general de ALCATRAZ.

Dentro de la industria española, Asturias siempre ha sido una de las Comunidades Autónomas con mayor relevancia, en gran parte debido a su industria siderúrgica y minera. No obstante, la industria asturiana lleva décadas sufriendo una fuerte transformación. Del declive del sector industrial tradicional (secundario) para dar paso a una industria más enfocada en el sector servicios a la doble transformación ecológica y digital (*Twin Transition*) concentrada en un fuerte proceso de descarbonización con las directrices europeas del Pacto Verde (*Green Deal*) con el ánimo de convertir a Europa en el primer continente climáticamente neutro. En el ámbito regional, la estrategia Asturias RIS3 para una especialización inteligente para el período 2014-2020 (se espera una actualización de esta estrategia en el 2021) estaba enfocada en 16 prioridades temáticas que responden a las fortalezas científico-tecnológicas e industriales de la región:

- **Materiales Avanzados y Sostenibles**
  Materiales para la Industria
  Materiales Sostenibles
  Nanomateriales y Grafeno

- **Nuevos Modelos de Producción**
  Fabricación Digital
  Fabricación Aditiva

- **Suministros y Tecnologías para Redes**
  Energía: Producción y Consumo
  Logística y Seguridad
  Gestión del Agua
  Análisis de Datos
  Sensores

- **Asturias Polo Industrial del Acero**
  Innovación Abierta en la Producción y Transformación del Acero
  Mercados de la Energía y Transporte

- **Mercados Agroalimentarios**
  Recursos Agroalimentarios
  Biotecnología en el Sector Lácteo

- **Envejecimiento Demográfico y Calidad de Vida**
  Biomedicina
  Polo de Salud

Se ha priorizado el desarrollo de nuevos materiales y modelos de producción y el enfoque en suministros y tecnologías para redes a la par que el refuerzo de industrias más tradicionalmente asociadas con Asturias, como son la industria del acero, la agroalimentaria y la salud. Por tanto, la reconstrucción industrial asturiana, a la espera de su nueva estrategia regional, llevará a una Asturias más verde, sostenible, saludable, digital, resiliente e inclusiva. El reto se centra, por tanto, en redirigir el modelo productivo de una forma inclusiva, sin dejar a nadie atrás. Los proyectos que lleven a cabo esta transformación tendrán que estar enfocados, desde el primer momento, en la colaboración multidisciplinar, público/privada y en la construcción conjunta de soluciones que generen valor añadido bruto y empleo sostenible y de calidad.

La relación entre los problemas matemáticos a tratar y los retos de la industria en los que se van a ejemplificar se muestra en la siguiente tabla:

Problemas matemáticos	Retos de la industria
Clasificación	Detección de anomalías
Optimización	Optimización de procesos
Simulación	Diseño de nuevos productos

A pesar de los posibles beneficios que puede aportar la computación cuántica en problemas de clasificación, optimización y simulación, hay que tener en cuenta que la aplicación de técnicas basadas en tecnologías cuánticas a casos reales es todavía escasa debido a que los ordenadores cuánticos comerciales tienen aún ciertas limitaciones, como una conectividad limitada, falta de memoria cuántica y un número bajo de qubits. Por tanto, este proyecto se centra en una comparación entre técnicas computacionales clásicas y cuánticas, sin presuponer una posible supremacía cuántica.

Detección de anomalías

La detección de anomalías es la identificación de desviaciones o discrepancias de una regla o de un uso, algo que no se espera en base a la información conocida. Es un campo de investigación tradicional basado en conceptos estadísticos principalmente, que siempre ha estado presente en la industria, ya que una detección temprana de anomalías evita desperdicios de materiales, ayuda a mejorar la calidad y reduce costes y paradas innecesarias de los sistemas de producción. Por tanto,

muchas empresas están interesadas en una detección temprana de anomalías desde un punto de vista económico.

Resulta interesante la aplicación de la detección de anomalías al **sector TIC**, al tratarse de uno de los sectores industriales predominantes en Asturias. En el sector TIC, la detección de anomalías se podría enmarcar como la **anticipación a ciberataques**. Se entiende por ciberataque cualquier maniobra ofensiva de explotación deliberada para tomar el control, desestabilizar o dañar un sistema informático y perjudicar a personas, instituciones o empresas. Los ciberataques constituyen ya uno de los principales riesgos para las compañías, de todo tipo de sectores, tamaños y localizaciones.

La continua complejidad de los procesos industriales requiere un análisis de datos avanzado que las técnicas clásicas pueden ser incapaces de resolver. Es por esto que la detección de anomalías aplicada a ejemplos industriales se plantea como un buen escenario sobre el que aplicar técnicas de computación cuántica con la esperanza que ofrezcan ventajas, como por ejemplo, en términos de velocidad.

Optimización de procesos

Un problema matemático de **optimización** llevado a la industria puede representarse con un reto de **optimización de procesos industriales**. La optimización de procesos ayuda a mejorar la calidad y la eficiencia de los mismos y/o a minimizar costes. Multitud de sectores de la industria buscan así mejorar su competitividad.

La **industria logística y del transporte** es un sector en el que la optimización de procesos cobra una importancia vital, ya que existen numerosas variables que deben ser continuamente equilibradas para lograr diferentes objetivos comerciales. Además, es un sector que se enfrenta a importantes retos en la próxima década, por lo que las empresas logísticas y de transporte tendrán que estar preparadas para los cambios apoyándose en el avance de la tecnología, las nuevas tendencias, las consecuencias de la pandemia por COVID-19 y la preservación del medio ambiente.

Las capacidades de la computación clásica no logran abordar en un tiempo aceptable la compleja red de variables que se interrelacionan entre sí en la industria logística y ofrecen soluciones que realmente no son óptimas.

Esta limitación de las técnicas clásicas nos hace plantearnos el uso de computación cuántica para un ejemplo de optimización de procesos logísticos, ya que ofrece la oportunidad de encontrar las mejores soluciones posibles especialmente en la cadena de suministro, con el análisis de múltiples variables heterogéneas que se relacionan entre sí.

Diseño de nuevos productos

Se considera el diseño de nuevos productos como la generación y desarrollo de ideas para crear un producto innovador y funcional, enfocado en solucionar una problemática concreta. Se plantea el uso de **simulación** para el **diseño de nuevos productos**, ya que la simulación de procesos con modelos matemáticos ayuda tanto a generar nuevas alternativas como a identificar aquéllas más adecuadas para conseguir los fines deseados, sin interrumpir líneas de producción y sin necesidad de diseñar ni construir prototipos, por lo que es un procedimiento útil y económico. Cuestiones como la evaluación

de nuevas alternativas para modificar una línea de producción antes de hacer la inversión, maximizar el consumo energético de una empresa y la adecuación de procesos para adaptarlos a una nueva composición de producto, se pueden abordar desde el ángulo de la simulación (ensayos *in silico*).

La crisis sanitaria derivada de la pandemia por COVID-19 ha puesto de manifiesto la necesidad de disponer de procesos de diseño de nuevos productos que sean flexibles, ágiles y altamente adaptables a cambios de demanda. Esto se ha visto, por ejemplo, en el diseño o identificación de fármacos para el tratamiento de la COVID-19 así como en el diseño de vacunas para su prevención. Esta conclusión, junto con el hecho de que en Asturias, el sector de la salud es uno de los campos industriales con más presencia, ha hecho que en ALCATRAZ nos decantemos por centrar el diseño de nuevos productos en el **sector farmacéutico**.
Una de las estrategias más utilizadas para la investigación de nuevos fármacos es la simulación, en concreto, la modelización de las funciones de onda que rigen el comportamiento de las partículas. Sin embargo, los ordenadores clásicos no se ajustan bien a este caso de uso. El volumen del problema crece hasta ser inabarcable en la mayoría de las situaciones reales. Por el contrario, los ordenadores cuánticos son una plataforma ideal para la modelización de las funciones de onda, que se comportan según los principios de la física cuántica por naturaleza.

En los tres casos expuestos, la definición final del caso de uso concreto se realizará en la segunda mitad del proyecto, una vez que se seleccionen los algoritmos cuánticos con mayor probabilidad de éxito para cada ejemplo.

T1.2: Análisis de los subproblemas

Definición de los problemas

Optimización

El rendimiento de las empresas de transporte y logística depende de la optimización de sus activos en las rutas que tienen que ejecutar diariamente. Es decir, la distribución de vehículos y las rutas planificadas influyen notablemente en los costes operacionales. Una mejora leve del rendimiento puede tener impacto a largo plazo en los beneficios y estabilidad de la empresa. Se define dicho subproblema de optimización en la tabla siguiente:

Parámetros de entrada y precondiciones	Objetivo	Oportunidades en computación cuántica
Mapa geográfico de objetivos de rutas (e.g. direcciones concretas, poblaciones). Conjunto disponible de vehículos, incluyendo sus características y capacidades.	Obtener la configuración óptima de vehículos y rutas para cumplir la planificación con el mínimo coste posible.	Este tipo de problemas de optimización combinatoria, similares al bien conocido problema del Traveling Salesman (TSP), suponen un problema engañosamente complicado en ordenadores clásicos. Encontrar la solución óptima se hace inabarcable

Entregas y rutas que deben ser cubiertas en un periodo concreto. Otras restricciones o costes específicos del negocio.		rápidamente con el crecimiento del tamaño del problema. De hecho, la versión de TSP para optimización es un problema NP-hard. Es decir, es un problema que es complejo de resolver y verificar en tiempo razonable (i.e. polinomial) en un ordenador clásico.

Clasificación

Normalmente las empresas necesitan servicios informáticos internos para desempeñar sus funciones de manera sostenible. En un número significativo de casos, deben tener también una presencia en Internet para sus clientes. Además, esa presencia debe proporcionar servicios de valor a los clientes, no suele ser aceptable limitarse simplemente a una presencia informativa. Todos estos servicios y sistemas dan lugar a una superficie de ataque que puede ser aprovechada por actores maliciosos. Un ataque puede provocar una disrupción inaceptable de las operaciones regulares, por lo que la implementación de medidas proactivas de ciberseguridad es una necesidad ineludible. Se define dicho subproblema de clasificación en la tabla siguiente:

Parámetros de entrada y precondiciones	**Objetivo**	**Oportunidades en computación cuántica**
Trazas de red que representan las comunicaciones en tiempo real de un servicio o aplicación.	Detectar la ocurrencia en tiempo real de ataques de denegación de servicio (o similares) a partir de la monitorización activa del tráfico de red.	Las referencias en el estado de la técnica apuntan a que los circuitos cuánticos pueden permitir el aprendizaje de “funciones cuánticas” que son difícilmente alcanzables con técnicas clásicas. Estas funciones cuánticas podrían tener un impacto significativo en el rendimiento de modelos de clasificación, especialmente si se adopta una aproximación híbrida que combine las ventajas de los bien conocidos modelos clásicos con las capacidades únicas de los circuitos cuánticos.

Simulación

La modelización de las funciones de onda que rigen el comportamiento de las partículas es la estrategia que se utiliza para la investigación de nuevos compuestos (e.g. medicamentos). El problema principal es que los ordenadores clásicos no se ajustan bien a este caso de uso. El volumen del problema crece hasta ser inabarcable en la mayoría de las situaciones reales. Por el contrario, los ordenadores cuánticos son una plataforma ideal para la modelización de las funciones de onda, que se comportan según los principios de la física cuántica por naturaleza. Se define dicho subproblema de simulación en la tabla siguiente:

Parámetros de entrada y precondiciones	**Objetivo**	**Oportunidades en computación cuántica**
Estructura de la molécula, incluyendo cualquier parámetro o condición inicial que sea necesario para modelar unívocamente el estado de la partícula (por ejemplo, las coordenadas nucleares).	Optimización de la geometría de moléculas simples, el cálculo de orbitales frontera en compuestos sencillos y simulaciones de dinámicas moleculares.	Como se comentaba anteriormente, la modelización de funciones de onda es una técnica básica para la simulación de procesos químicos que no se ajusta bien a las características de los procesadores clásicos. Sin embargo, los ordenadores cuánticos se rigen por los mismos principios de superposición, por lo que son un buen encaje para representar funciones de onda.

Identificación de técnicas

A lo largo de este apartado, se llevará a cabo un análisis acotado de las diferentes técnicas que se consideran como posibles soluciones para abordar las tres temáticas previamente seleccionadas y descritas. Dada la estructura propia del proyecto, se dividirá a su vez este apartado en dos, donde en el primero se tratan las técnicas de computación clásica, y en el segundo, las relativas a la computación cuántica.

Computación clásica

Para facilitar la estructuración y visualización de las técnicas consideradas, se presentan las distintas técnicas detectadas en la siguiente tabla, asociada al reto en cuestión, y con la descripción breve pertinente.

Subproblema matemático	**Técnica**	**Descripción**
Optimización	Algoritmos sin restricciones	Se tratan de algoritmos de optimización cuya principal característica está asociada a que las únicas restricciones

		propias del problema son que las variables involucradas sean números reales. Dentro de esta tipología, se pueden distinguir distintos tipos, tales como el bracketing, el descenso local, los métodos directos, los algoritmos estocásticos y los algoritmos poblacionales. En caso de no existir dichas restricciones adicionales al problema, podrá ser una técnica que se adapte al problema a tratar.
	Algoritmos con restricciones	Se tratan de algoritmos de optimización que, en contraste con los algoritmos sin restricciones, no buscan la solución en todo el espacio real, sino que se ven limitadas a subespacios de éste dados por restricciones adicionales a las variables en estudio. Al igual que en el caso previo, se pueden encontrar diferentes categorías, tales como la optimización con restricciones lineales, así como la optimización multi-objetivo. En caso de existir dicho tipo de restricciones en el problema planteado, deberán considerarse dichas técnicas prioritariamente con respecto a las previas.
	Algoritmos para problemas indefinidos	Se tratan de algoritmos de optimización para situaciones donde la función objetivo o las restricciones asociadas no están definidas del todo,

		pudiendo ocurrir bien por posibles aproximaciones sobre el propio modelo, así como a través de fluctuaciones entre los parámetros. En caso de que el problema de optimización tratado se encuentre en dicha situación, podrá recurrirse a este tipo de algoritmos.
	Optimización discreta	Se tratan de algoritmos de optimización cuya principal característica diferenciadora es la utilización de variables cuyos valores no son continuos sino discretos. Se contempla como posible alternativa en caso de ser necesaria, si bien a priori no parece ser una rama de la optimización necesaria para el modelado de este proyecto.
Clasificación	Basados en modelos probabilísticos	Algoritmos de aprendizaje automático supervisado de clasificación que utilizan como base la inferencia para identificar la clase más adecuada para los nuevos individuos a categorizar. Entre los diferentes modelos dentro de esta categoría, podemos encontrarnos tanto la clásica técnica conocida como regresión logística (modelo inspirado en la regresión lineal utilizada para problemas de regresión, y basada en la función logística), así como el modelo Naive Bayes (basado en el teorema de Bayes, especialmente útil en espacios de alta

		dimensionalidad).
	Árboles de decisión	Modelos basados en sistemas de concatenación de diferentes decisiones lógicas y que permiten determinar con ellas la salida del modelo, a través de un recorrido basado en el desplazamiento entre los nodos y ejes que los componen. Se pueden identificar diferentes tipos de árboles, si bien una de las aplicaciones más habituales son los conocidos como Random Forest, basados en la combinación de un conjunto de árboles individuales, donde los valores predichos finales son obtenidos a través de la agregación de los resultados individuales de cada uno de ellos.
	Basados en instancias	Modelos cuya base principal es la agrupación de los distintos individuos con respecto a su similitud frente al resto de instancias del conjunto de datos. Como es evidente, son modelos dependientes de la distancia o similitud utilizada como base del modelo, pudiendo usarse de forma habitual cuatro de las distancias más usuales: euclídea, Manhattan, Minkowski o Hamming. Entre dichos modelos, el más conocido es el de los k-vecinos más cercanos, donde se busca categorizar nuevos individuos basándose en la clasificación

		conocida de los k individuos más cercanos, según la distancia seleccionada.
	Máquinas de vector soporte	Modelos de aprendizaje automático supervisado que son aplicables tanto a casos de clasificación como de regresión, si bien su concepción inicial se asocia a clasificación binaria. Buscan la separación de las diferentes categorías a través de hiperplanos que separen adecuadamente dichas clases. Entre las diferentes características de estos modelos, radica la variedad de kernels disponibles para su aplicación, pudiendo así generar resultados dispares.
	Métodos de conjuntos	Se tratan de métodos que buscan la combinación de diferentes tipos de clasificadores con el objetivo de mejorar la precisión de dicha clasificación. Se pueden contemplar diferentes tipos de dicha clase de métodos, tales como los de bagging o de boosting. Un ejemplo de estos modelos de bagging es el previamente mencionado en el apartado de árboles, en particular, los Random Forest. En el caso de los modelos de boosting, modelos como AdaBoost o XGBoost forman parte de los que mejores resultados suelen presentar.
Simulación	Teoría del funcional de la densidad (DFT)	Se trata de una técnica más eficiente que los métodos Hartree-Fock y, a la vez, bastante precisa. Se basa en el

		teorema de Hohenberg-Kohn, que muestra que es posible usar simplemente la densidad electrónica para calcular la energía del estado fundamental E0 de una molécula. Esto reduce significativamente la complejidad del problema al considerar que la densidad electrónica es función de sólo tres coordenadas y es independiente del número de electrones.

Computación cuántica

De manera similar a la sección anterior de computación clásica, se presenta a continuación una tabla que contiene descripciones breves de las técnicas de computación cuántica identificadas como posibles opciones para acometer los retos.
Las técnicas serán descritas con mayor detalle en tareas posteriores. En estas tareas se evaluarán sus características y adecuación, para finalmente seleccionar el conjunto que se utilizará para la implementación de soluciones y pruebas de concepto finales.

Subproblema matemático	Técnica	Descripción
Optimización	*Quantum Approximate Optimization Algorithm* (QAOA)	El algoritmo QAOA propone definir los problemas de optimización en términos de un Hamiltoniano que representa la evolución de la "energía" de un circuito a través del tiempo. Este Hamiltoniano, denominado "de coste" se combina con otro Hamiltoniano "de mezcla" en una secuencia de capas cuánticas repetibles. Los parámetros se optimizan usando técnicas clásicas (e.g. *gradient descent*). El estado de menor energía del circuito QAOA representa una solución óptima al problema de

		optimización modelado
	Feedback-Based Quantum Optimization (FALQON)	FALQON es una alternativa a QAOA que evita el uso de técnicas clásicas de optimización. A diferencia de QAOA, FALQON permite asegurar la convergencia. Sin embargo, las necesidades de computación en FALQON escalan peor con el tamaño del problema que en el caso de QAOA. Algunas referencias proponen la combinación de QAOA y FALQON para obtener un mayor rendimiento.
Clasificación	*Quantum Convolutional Neural Networks* (QCNN)	El modelo QCNN propone una visión puramente basada en puertas cuánticas de la arquitectura de las redes convolucionales clásicas. En las redes QCNN, las capas convolucionales y de pooling se implementan respectivamente con combinaciones de puertas parametrizables y medidas para reducción de la dimensionalidad.
	Quanvolutional Neural Networks (QNN)	El modelo QNN se diferencia del anterior QCNN en que propone una aproximación híbrida que conserva las capas convolucionales y de *pooling* clásicas. La contribución principal de las redes QNN es el añadido de un nuevo tipo de capa denominada quanvolutional. Esta capa se basa en un

		circuito aleatorio que permite la aplicación de un kernel cuántico que está fuera del alcance de las capas implementadas con computación clásica.
	Quantum Transfer Learning	El Transfer Learning es una técnica basada en la reutilización de redes con rendimiento notable que han sido previamente entrenadas para problemas generalistas. Las últimas capas de la red se sustituyen por capas de propósito específico entrenadas para un problema particular. El *Quantum Transfer Learning* es la aplicación de estos mismos principios, que han demostrado buenos resultados, a la computación cuántica. Es decir, las capas de propósito específico se implementan sobre circuitos cuánticos (*Quantum Processing Units* o QPU).
	Data Re-uploading Universal Quantum Classifier	Una arquitectura de red basada en la aproximación innovadora de utilizar un solo qubit. Además, los nodos de entrada se conectan de manera recurrente en todas las capas de la red, de ahí el clasificador de "*re-uploading*". Presentan un rendimiento notablemente bueno, especialmente cuando se tiene en cuenta que solo se necesita un qubit.
	Variational Quantum Classifiers	La arquitectura de esta red está compuesta por capas

		reutilizables que contienen rotaciones arbitrarias, además de otras puertas tales como CNOT para la creación de entrelazamientos. Los parámetros de rotación se someten al proceso de entrenamiento de la red. Los entrelazamientos en este caso son un componente básico para la modelización de funciones no lineales y el aprovechamiento de las características de los ordenadores cuánticos.
Simulación	*Variational Quantum Eigensolver* (VQE)	Una técnica híbrida clásico-cuántica que permite obtener los eigenvalues de un circuito cuántico variacional. Concretamente, tiene aplicación extensa en el dominio de la simulación química basada en computación cuántica. Por ejemplo, para el cálculo de ground states (i.e. estados de menor energía) o la modelización de la estructura molecular. Además, podría tener también aplicación en el contexto de los problemas de clasificación.

PT2: Análisis, diseño y desarrollo de algoritmos basados en computación clásica

Este paquete de trabajo se organiza en torno a cuatro tareas, de las cuales dos han sido iniciadas y finalizadas en su totalidad en esta anualidad, y una tercera ha sido iniciada con previsión de finalización en el año 2022. La restante, se iniciará y finalizará a lo largo de la próxima anualidad.

- T2.1: Análisis del estado de la técnica en algoritmos de computación clásica relacionados con clasificación, optimización y simulación
- T2.2: Identificación y selección de algoritmos de computación clásica relacionados con clasificación, optimización y simulación
- T2.3: Diseño e implementación de algoritmos de computación clásica relacionados con clasificación, optimización y simulación (iniciada, pero no finalizada)

A continuación, se describen las acciones desarrolladas en cada una de las tareas y los principales resultados alcanzados en cada una de ellas. En el caso de la tarea iniciada, pero no finalizada, se dará una breve descripción de los desarrollos realizados hasta el momento y que se presentarán de forma finalizada en la próxima anualidad.

T2.1: Análisis del estado de la técnica en algoritmos de computación clásica relacionados con clasificación, optimización y simulación

En esta tarea se presenta el estado actual de la técnica en los campos de la clasificación y optimización desde el punto de vista de la computación clásica. Se presentará cada uno de estos campos incluyendo algunos conceptos básicos cuando sean pertinentes, los algoritmos más utilizados y algunos ámbitos de aplicación.

Clasificación

La clasificación es un área dentro de la inteligencia artificial y más concretamente dentro del aprendizaje automático o *machine learning* cuya tarea es analizar las características de diferentes elementos de un conjunto de datos y etiquetarlos dentro de grupos con características similares, es decir, clasificarlos. La clasificación se puede tipificar según los siguientes factores:
Según el número de posibles clases de la variable objetivo:

- Binaria: sólo existen dos posibles categorías para la variable objetivo.
- Multiclase: el número de posibles categorías es mayor de dos.
- Multietiqueta: la variable objetivo puede pertenecer a más de una categoría.

Según la respuesta provista por el algoritmo:

- Categoría concreta: el algoritmo de clasificación produce como resultado la categoría concreta a la que pertenece cada elemento.
- Probabilidad de pertenecer a cada posible categoría: el algoritmo de clasificación produce como resultado una lista de porcentajes correspondiente a la probabilidad de que un elemento pertenezca a cada una de las posibles clases.

Según la existencia o no de datos previos sobre los que aprender:

- Clasificación supervisada
- Clasificación no supervisada

En esta tarea se va a hacer hincapié en esta última forma de tipificación ya que es la que tiene más impacto en cómo abordar el problema.

Clasificación supervisada

En la clasificación supervisada existe un conjunto de datos generalmente denominado “conjunto de entrenamiento” en el cual cada uno de los elementos ya está etiquetado dentro de una categoría. Este conjunto de entrenamiento es entonces utilizado por un modelo para establecer relaciones entre los

elementos de la misma categoría de manera que cuando se le presenten nuevos datos sin etiquetar o "datos de test", éste sea capaz de clasificar sus elementos en la categoría correcta.
Existen multitud de algoritmos de clasificación y se prevé que este número siga aumentando a medida que se descubren nuevas formas de abordar este tipo de problemas. Es por ello que no se van a exponer aquí todos los algoritmos existentes, pero se tratará de presentar aquellos que abarquen la mayor parte de los problemas relacionados con esta área.
A continuación, se presentarán los tipos de algoritmos de clasificación supervisada más utilizados y presentes en la literatura en el momento de la redacción de este documento.

Basados en modelos probabilísticos

Los algoritmos de clasificación basados en modelos probabilísticos utilizan la inferencia para encontrar la mejor clase para un elemento dado. Además de simplemente asignar la clase correspondiente como otros algoritmos de clasificación, los algoritmos de clasificación probabilística generarán otra lista de valores correspondientes a la probabilidad de que el elemento sea miembro de cada una de las clases posibles. La clase normalmente se selecciona como la mejor clase. En general, los algoritmos de clasificación probabilística tienen algunas ventajas sobre los clasificadores no probabilísticos: primero, puede generar un valor de confianza (es decir, probabilidad) asociado con su etiqueta de clase seleccionada, y por lo tanto puede abstenerse si su confianza de elegir cualquier resultado en particular es demasiado baja. Segundo, los clasificadores probabilísticos pueden incorporarse de manera más efectiva en tareas de aprendizaje automático más grandes, de una manera que parcial o por completo evita el problema de la propagación de errores. Dentro de este tipo de algoritmos podemos destacar Naive Bayes y la Regresión logística.
El clasificador Naive Bayes se basa en el teorema de Bayes y es particularmente adecuado cuando la dimensionalidad de los datos es alta. A pesar de su simplicidad, el clasificador Naive Bayes a menudo puede lograr un rendimiento comparable con algunos métodos de clasificación sofisticados que se presentarán más adelante, como los árboles de decisión y algunos clasificadores basados en redes neuronales. Este clasificador también ha exhibido una alta precisión. y velocidad cuando se aplica a grandes conjuntos de datos.
La regresión logística, al igual que Naive Bayes es un modelo probabilístico que trata de estimar $p(Y|X)$, es decir, la probabilidad de que se dé cierta clase dado un conjunto de valores de atributos de los datos. Generalmente la regresión logística se emplea para la clasificación binaria por lo que en lo referente a este método se centrará para el caso en que Y pueda tomar sólo 2 posibles valores, es decir, $Y \in \{0,1\}$. Este método emplea lo que se conoce como función logística o función sigmoid.

Árboles de decisión

Otra de las técnicas características de clasificación es el árbol de decisión. Este algoritmo se representa como un conjunto de nodos y ramas que los conectan. Cada nodo interno representa una decisión del algoritmo y una partición del árbol hacia la siguiente decisión hasta llegar a los nodos finales u hojas que representan la predicción final sobre la clase de la variable objetivo. Cada una de las ramas puede entenderse como una expresión lógica que dependiendo de si el resultado de la misma es verdadera o falsa el algoritmo irá en la dirección de uno de los nodos hijos de manera que el árbol en su conjunto se trate de una serie de expresiones lógicas unidas mediante un operador "and". Lo más común es que cada rama haga referencia a un atributo concreto del conjunto de datos, aunque existen estudios sobre árboles donde cada decisión puede tener en cuenta más de un atributo que han dado buenos resultados.

Los árboles de decisión son típicamente binarios, es decir, que en cada decisión solo existen dos posibles respuestas que dividen el espacio de valores de cada atributo en dos, aunque también existen versiones multi-decisión donde se debe tener especial cuidado ya que en algunas ocasiones demasiadas opciones pueden llevar a pérdidas considerables en cuanto a la precisión.
Uno de los principales retos que suponen los árboles de decisión es que, aunque no es complicado construir un modelo que consiga clasificar un conjunto de datos en las clases correctas, sí lo es construir un modelo que lo haga en un tiempo óptimo.
Encontrar el árbol mínimo que sea consistente con el conjunto de datos de entrenamiento para este tipo de modelos es un problema considerado NP-hard y para el caso concreto de árboles binarios es considerado NP-completo, es decir, que desde el punto de vista de la complejidad computacional se trata de problemas de complejidad al menos similar a los problemas no deterministas que pueden resolverse en tiempo polinómico.
El proceso de aprender sobre la estructura de un árbol de decisión y de construir un clasificador se conoce como inducción del árbol de decisión. Como se ha comentado antes este proceso es difícil de realizar de manera óptima por lo que requiere del uso de heurísticos para mejorarlo. Dentro de la inducción de árboles de decisión existen dos tipos que son los de arriba hacia abajo que emplean la técnica de "divide y vencerás" y los de abajo hacia arriba con una clara preferencia por el primer grupo en la literatura científica.

Máquinas vector soporte

Las máquinas de vector soporte surgen como una forma de reducir el problema del sobreajuste de los modelos de clasificación que provoca que obtengan buenos resultados para predecir conjuntos de entrenamiento pero no para nuevos conjuntos de datos. Las máquinas de vector soporte se basan en el concepto de espacios linealmente divisibles mediante hiperplanos que a su vez se basan en el uso de clasificadores de máximo margen.
Los clasificadores de máximo margen son clasificadores binarios que funcionan cuando hay una frontera lineal que separa perfectamente los datos de entrenamiento pertenecientes a una clase de los de otra.
Existen múltiples instancias que equidistan del hiperplano con el máximo margen y cuya distancia es precisamente el margen. Estas instancias son los vectores soporte que dan nombre a esta técnica.
Las máquinas vector soporte también permiten trabajar con datos no lineales. Para ello se realizan una serie de transformaciones sobre los datos de entrenamiento mediante una función de mapeo $\Phi(z)$ y posteriormente se emplea lo que se conoce como funciones kernel:
Algunos ejemplos de funciones kernel son:

- Función de base radial gaussiana
- Kernel polinómico
- Tangente polinómica

Basados en instancias

La técnica de clasificación basada en instancias consiste en agrupar instancias similares del conjunto de datos en las mismas clases. Esta técnica también se la conoce como aprendizaje perezoso en algunos casos ya que algunos de los métodos utilizados esperan a tener algún conocimiento de los datos de test para crear un modelo específico para los datos con los que se está lidiando en el momento en vez de generar previamente un modelo que intente generalizar la salida para cualquier posible dato de test.

Al igual que otros tipos de algoritmos de clasificación, los algoritmos basados en instancias producen como resultado un *concept description* que es una función que asocia instancias de los datos a categorías o clases concretas. Sin embargo, estos además almacenan conjuntos de datos y en algunas ocasiones incluso información referente al rendimiento que han tenido dichas instancias en clasificaciones pasadas y a medida que se vayan realizando tareas de clasificación, este conjunto de datos se irá modificando para amoldarse a nuevos datos. Los algoritmos basados en instancias poseen 3 componentes principales:

- Función de distancia o similitud: esta se encarga de establecer las similitudes entre elementos del conjunto de datos de entrenamiento y/o entre datos del conjunto de entrenamiento y del conjunto de test. Existen fundamentalmente 4 formas de medir la distancia entre elementos que son:
  - Distancia Euclidea
  - Distancia de Manhattan
  - Distancia de Minkowski
  - Distancia de Hamming
- Función de clasificación: esta función realiza la clasificación de una instancia del conjunto de test a partir de las relaciones establecidas en la función de similitud anterior.
- Actualizador del concept description: este componente se encarga de modificar el conjunto de datos almacenado por el modelo en cada clasificación dependiendo del rendimiento obtenido en la última clasificación.

El algoritmo más utilizado y conocido dentro de los clasificadores basados en instancias es el *Nearest Neighbor* o k-Vecinos más Cercanos. Tal es así que incluso en parte de la literatura relacionada con este tipo de clasificación se consideraba a *Nearest Neighbor* y la clasificación basada en instancias como un mismo concepto.
Este algoritmo trata de buscar, de entre las instancias existentes, un número k de elementos con características similares a la instancia de test de manera que esta tome el valor de la clase que compartan las demás instancias.

<u>Redes neuronales</u>

Las redes neuronales son modelos estadísticos adaptativos que se asemejan a la estructura de las neuronas del cerebro humano. Estas reciben una señal de entrada desde unas conexiones denominadas dendritas y generan una señal de salida que envían a través de otra conexión llamada axón que a su vez interactúa con las dendritas de otras neuronas mediante pesos sinápticos.
La estructura de las redes neuronales se basa en el uso de nodos que son núcleos computacionales denominados neuronas unidos entre sí en forma de un grafo que puede ser dirigido o no dirigido. De manera similar al proceso biológico, cada una de estas neuronas recibe datos de entrada de un conjunto de unidades de entrada y genera una señal de salida que envía a través de un conjunto de unidades de salida habiendo otro conjunto de unidades intermedias encargadas de procesar la lógica de las operaciones.
El modelo matemático de una neurona transforma un conjunto de entradas en una sola salida mediante la composición de dos funciones:

- Función de red: esta utiliza los parámetros o pesos de la neurona en cuestión para resumir el conjunto de datos de entrada x en una valor de red v. Las funciones de red más utilizadas son:
  - Suma de pesos
  - Distancia
  - Kernel definido por el usuario

- Función de activación: esta transforma el valor de red en el valor de salida de la neurona. De manera similar a como ocurre con las neuronas biológicas, esta función, dependiendo de su lógica interna, generará la señal de salida o la inhibirá. Las funciones de activación más utilizadas son:
    - Función lineal
    - Función de umbral
    - Función sigmoidea
    - Función tangente hiperbólica

Las redes neuronales pueden tener diferentes topologías dependiendo de las suposiciones del modelo en cuestión, aunque para problemas de clasificación requieren tener al menos una capa o nivel de entrada para introducir los datos a la red y otro de salida para devolver la respuesta. Las tipologías más comunes son las redes multicapa, las redes modulares y las redes con retroalimentación, aunque es importante remarcar que existen también redes neuronales monocapa.

Métodos de conjuntos

Los métodos de conjuntos son aquellos que combinan el uso de diferentes tipos de clasificadores para mejorar la precisión de un problema de clasificación. Poseen un gran potencial para obtener buenos resultados en multitud de problemas debido a la diversidad de aproximaciones proporcionadas por cada uno de los modelos utilizados. Generalmente si se posee un número de clasificadores para un problema dado y uno de ellos obtiene la mejor precisión, entonces se suele utilizar únicamente éste como modelo para futuros datos. Sin embargo, se ha probado que debido a la incertidumbre de los datos con los que se va a trabajar en el futuro, el uso de conjuntos de clasificadores provee una mayor seguridad en cuanto a mantener o incluso mejorar la precisión final.
Debido al éxito que han tenido este tipo de modelos conjuntos, se han implementado numerosas versiones de conjuntos de modelos, pero entre ellos destacan el *bagging* y el *boosting*.
El método de *bagging* consiste en entrenar un conjunto de algoritmos de clasificación a partir de nuevos conjuntos de datos generados a partir de los originales utilizando una técnica conocida como bootstrap. Una vez se han entrenado estos algoritmos, se combinan utilizando un sistema de voto por mayoría para generar el conjunto de modelos.
Uno de los mayores ejemplos de este método es el algoritmo de *Random Forest*. Este algoritmo utiliza árboles de decisión como algoritmos de clasificación para generar el conjunto de modelos los cuales al final del proceso deciden por votación cuál de las soluciones propuestas por los diferentes modelos es la salida final del conjunto. Además, incorpora un componente de aleatoriedad a la hora de generar los nuevos conjuntos de datos para aumentar la diversidad de estos.
El método de *boosting* consiste en construir un conjunto de modelos a partir de un grupo de clasificadores de manera gradual mediante la asignación de pesos en función del éxito de cada uno de los clasificadores a la hora de identificar instancias. En la primera iteración, el método de *boosting* utiliza uno de sus clasificadores para clasificar el conjunto de datos de entrenamiento y asigna un peso a las instancias mal clasificadas. En las sucesivas iteraciones se irán utilizando los demás clasificadores focalizándose en las instancias con mayor peso e irán incrementando o disminuyendo el peso de las instancias mal clasificadas de manera que el conjunto global irá aprendiendo de las iteraciones anteriores generando un modelo global más preciso.

Clasificación no supervisada

En la clasificación no supervisada no existe un conjunto de entrenamiento con el que un modelo pueda aprender. Esto puede deberse a que directamente no existe o que el coste de etiquetar manualmente cada uno de los elementos de un conjunto de datos es demasiado elevado. Es por ello que este tipo de clasificación requiere una aproximación diferente. Se debe realizar un proceso previo a la clasificación supervisada conocido como clustering en el que se agrupan objetos con características similares en grupos de manera que se disponga un conjunto de datos con los que trabajar (Bandyopadhyay & Saha, 1998).

Ámbitos de uso más frecuente

La técnica de clasificación es una técnica muy estudiada en diferentes campos como la minería de datos y el aprendizaje automático. Los usos de esta técnica incluyen una gran variedad de problemas como el tratamiento de textos relacionado con la detección de spam, para datos médicos como predicción de enfermedades, para el ámbito político como la predicción de votos, el tratamiento de imágenes para reconocimiento facial e incluso en el ámbito de la industria esta técnica es ampliamente utilizada para tareas como identificación de fraudes, predicción del comportamiento del mercado o estimación de fallos en maquinaria.

Optimización

La optimización en el ámbito de la ingeniería es el proceso de encontrar el punto para el que una función denominada función objetivo maximiza o minimiza su resultado. Conocer el comportamiento de la función y cómo afectan sus parámetros al resultado final es fundamental y para ello es necesario definir los parámetros, constantes, objetivos y restricciones que constituyen el problema. En esta sección se dividirá en 3 secciones principales: la primera consistirá en los elementos fundamentales que constituyen los problemas de optimización, la segunda hará un resumen de los principales algoritmos y la última parte comentará algunos de los ámbitos de uso.

Elementos fundamentales

En esta sección se describen brevemente conceptos fundamentales de la optimización que han de tenerse en cuenta para el resto de las secciones. Aquí se describirá el concepto de derivada, puntos críticos, restricciones y la definición básica de un modelo de optimización.

Derivada

Uno de los principales conceptos en el los que se fundamenta la optimización es el concepto de derivada de una función como la pendiente de la recta tangente a dicha función. Con esto es posible realizar una aproximación lineal del valor de dicha función en dicho punto. Sin embargo, esta es la expresión para un problema univariable. Para funciones multi-variable se emplean conceptos como el gradiente, la matriz hessiana y la derivada direccional.

Punto críticos

Los puntos críticos atendiendo a la anterior definición de la derivada constituyen los puntos del dominio de la función en los cuales el valor de la derivada es 0. Dependiendo de si se trata de encontrar el máximo o el mínimo de una función se entiende que los puntos críticos de esa función

son máximos o mínimos respectivamente. Estos a su vez pueden ser máximos/mínimos globales o locales dependiendo si de entre ellos es el valor que maximiza/minimiza el resultado para el dominio de la función o solo para un intervalo del mismo.

Restricciones

Las restricciones son una parte fundamental de los problemas de optimización ya que éstas limitan el espacio de posibles soluciones y evitan que el problema tenga una solución trivial o imposible que no se corresponde con la realidad.

Definición del problema de optimización

El diseño básico de un problema de optimización es:

maximizar, f(x) sujeto a: x∈X

Donde f(x) es la función objetivo a optimizar, en este caso el objetivo es maximizar su resultado, y x∈X es la restricción impuesta al problema. En el caso de que se trate de un problema sin restricciones impuestas se entiende X es el espacio de los números reales para el espacio n-dimensional del problema. Definir correctamente el problema a tratar es indispensable para reducir el tiempo necesario para encontrar la solución.

Algoritmos

Existen multitud de métodos y algoritmos para identificar el óptimo de un problema. En esta sección se comentarán algunos de ellos.

Algoritmos sin restricciones

Este conjunto de algoritmos de optimización se caracteriza por tener como restricciones del problema únicamente que las variables involucradas se encuentran dentro del espacio de los números reales.

Bracketing

Bracketing es el proceso de identificar un intervalo en el que se encuentre un mínimo o máximo local e ir acotando dicho intervalo hasta encontrar dicho punto crítico. La mayoría de los algoritmos de *bracketing* se basan en el concepto de función unimodelo. Una función unimodal es aquella en la que existe un único punto tal que la función es monótonamente creciente o decreciente, lo cual indica que el mínimo/máximo global se encuentra en dicho punto.

Descenso local

El algoritmo de descenso local consiste en determinar un punto inicial y a continuación generar una lista de puntos para los que el valor de la función se minimiza o maximiza respecto a éste. El proceso iterativo en términos generales para encontrar dicho punto crítico es el siguiente:

- Comprobar que el elemento elegido en la iteración k cumple el criterio de parada. Si lo cumple, el algoritmo termina y si no prosigue al siguiente paso.

- Determinar la dirección de descenso usando información local como el gradiente o la matriz Hessiana.
- Determinar el nivel de descenso o ratio de aprendizaje.
- Calcular el nuevo valor de referencia k+1.

La elección del punto inicial determinará el éxito del algoritmo para encontrar el punto crítico deseado. Para determinar si el algoritmo ha terminado existen diferentes criterios de convergencia. Los más comunes son:

- Número de iteraciones o tiempo de ejecución: si se alcanza el número máximo de iteraciones o el tiempo máximo de ejecución establecido por el experto entonces se termina la ejecución.
- Valor absoluto de mejora: se calcula en valor absoluto la mejora obtenida en la iteración actual y se compara con la de la anterior y si el valor obtenido es inferior a un valor predefinido por el experto, el algoritmo terminará.
- Valor relativo de mejora: al igual que en el caso anterior, se calcula la mejora obtenida en esta iteración respecto de la anterior, pero se compara con el valor v con respecto al valor de la iteración actual.

Para determinar la dirección de descenso se han desarrollado diversos métodos, pero entre ellos destacan fundamentalmente dos aproximaciones: los métodos de primer orden y los métodos de segundo orden.
Los métodos de primer orden se basan en la información obtenida en el gradiente para determinar la dirección de búsqueda del punto óptimo. Este método también se lo conoce como descenso de gradiente el cual se basa en el método de descenso más empinado.
Los métodos de segundo orden se basan en el uso de la segunda derivada y de la matriz Hessiana para determinar la dirección. De entre ellos el que más destaca es el método de Newton.
Al igual que ocurre con los métodos de primer orden existen implementaciones de este método que optimizan la convergencia del mismo así como minimizar algunos de los errores derivados de las consideraciones mencionadas anteriormente como el método de la secante o algunos métodos quasi newtonianos.

Métodos directos

Los métodos directos se caracterizan por depender únicamente de la función objetivo y no de información obtenida como en otros métodos anteriores para guiarse hacia el punto crítico o para saber cuándo lo han alcanzado. De entre los diferentes métodos directos existentes destacan la búsqueda cíclica de coordenadas, el método de Powell y la búsqueda generalizada de patrones.
La búsqueda cíclica de coordenadas. Ésta simplemente va alternando entre las diferentes posibles direcciones dentro de un mismo eje cada vez mientras el resto de los valores permanecen iguales.
Método de Powell. Este consiste en dividir el problema de maximización/minimización de tamaño N en N sub-problemas de optimización independientes y unidimensionales para los cuales se aplica una búsqueda binaria como la búsqueda de la sección dorada mencionada anteriormente.
Búsqueda generalizada de patrones. Trata de construir un patrón a partir de un conjunto de direcciones, las cuales pueden ser arbitrarias y no pertenecer a las direcciones de las coordenadas y un punto con un tamaño de salto dado. Para que este método pueda converger se tiene que cumplir que el conjunto de direcciones sea un sistema generador, es decir que se pueda construir cualquier punto del espacio n-dimensional de los números reales usando una combinación lineal de las direcciones del conjunto.

Algoritmos estocásticos

Los algoritmos de optimización estocásticos emplean cierto grado de aleatoriedad para incrementar las posibilidades de encontrar el valor óptimo de un problema. Además, el uso de valores aleatorios puede facilitar encontrar un máximo/mínimo global en vez local. Para ello generalmente se emplean generadores de números aleatorios, los cuales generan números aparentemente aleatorios a partir de algoritmos y distribuciones estadísticas o a elementos de hardware.
Existen múltiples implementaciones de algoritmos estocásticos, incluidas versiones de algoritmos mencionados anteriormente a los que se les ha añadido cierto componente de aleatoriedad como el descenso de gradiente ruidoso o métodos directos con mallas adaptativas.
Otro algoritmo interesante es el conocido como enfriamiento simulado o recocido simulado. Éste toma inspiración del proceso de recocido en la metalurgia el cual consiste en calentar un metal hasta una temperatura durante un tiempo prefijado para, a continuación, enfriarlo lentamente haciendo que este sea más fácil de trabajar. En cada iteración, el algoritmo evalúa otros estados vecinos del estado actual y decide si cambiar a otro estado según una probabilidad calculada.

Algoritmos poblacionales

Los algoritmos poblacionales son aquellos que emplean un conjunto de puntos de decisión en lugar de uno solo como en los métodos mencionados anteriormente. Si este conjunto de puntos de decisión se encuentra distribuido a lo largo del espacio del problema evita que el algoritmo se quede estancado en máximos/mínimos locales por lo que facilita encontrar el máximo/mínimo global.
La parte inicial de este tipo de algoritmos es parecida a la empleada en los métodos de descenso local mencionados anteriormente en los que se debe escoger un punto de decisión inicial salvo que en este caso debe escogerse un conjunto de ellos. Existen varias formas de inicializar este conjunto de puntos, pero los más comunes son:

- Utilizando un hiperrectángulo en el que los elementos de cada coordenada provengan de una distribución uniforme.
- Utilizando una distribución normal multivariante alrededor de la zona de interés. Esta aproximación cubre todo el espacio de búsqueda debido a que las matrices de covarianza son generalmente diagonales.
- Utilizando una distribución de Cauchy ya que ésta no posee una varianza definida y por lo tanto puede cubrir un espacio mayor a diferencia de las distribuciones uniforme y normal que por sus características, están limitadas a un espacio concentrado.

De entre los algoritmos poblacionales más conocidos se encuentran los algoritmos genéticos, algoritmos basados en la evolución diferencial.
Los algoritmos genéticos se inspiran en la evolución biológica en la que individuos con una genética que les proporcione mejores capacidades es más posible que sobreviva y que por lo tanto pase sus genes a las siguientes generaciones. El punto de diseño es representado como un cromosoma y en cada generación el algoritmo modifica el valor de optimización mediante dos operaciones conocidas como crossover y mutación.
Los algoritmos basados en la evolución diferencial tratan de mejorar cada individuo del conjunto de puntos de decisión mediante recombinaciones de otros elementos del conjunto. Estos algoritmos están parametrizados por una probabilidad de cruce y un peso diferencial que puede tomar valores entre 0 y 2 pero que generalmente se usan valores entre 0.4 y 1.

<u>Algoritmos con restricciones</u>

Este conjunto de algoritmos, a diferencia de los mostrados anteriormente, están acotados a un subespacio definido por una serie de expresiones o restricciones.

Optimización con restricciones lineales

Este tipo de problemas de optimización se caracterizan por tener como objetivo una función de tipo lineal, así como una serie de restricciones también lineales. Esto puede deberse a que la naturaleza del problema es directamente lineal o que se ha llevado una aproximación lineal de uno que no lo es. Este tipo de problemas puede expresarse de diferentes formas: general, estándar o de igualdad.
Dentro de este tipo de algoritmos, el que más destaca es el algoritmo Simplex. Este resuelve problemas de optimización lineal buscando el óptimo entre los puntos extremos de la región factible. Gráficamente la región factible puede representarse como el espacio formado por la intersección de todos los planos generados por las restricciones del problema. Los vértices de este espacio, así como los puntos a lo largo de sus lados son los posibles valores óptimos del problema ya que los puntos interiores se pueden optimizar al desplazarse a lo largo los ejes.
Matemáticamente el método Simplex genera lo que se conoce como un sistema inicial en el que se toma el modelo en cuestión en forma de igualdad y se genera un sistema de ecuaciones formado por la función objetivo igualada a 0 junto con las restricciones.

Optimización multi-objetivo

La optimización multi-objetivo u optimización de vectores trata aquellos problemas en los que se deben optimizar múltiples objetivos al mismo tiempo.
Un concepto fundamental para la optimización multi-objetivo es el de la eficiencia de Pareto o la optimalidad de Pareto. Se dice que un diseño de modelo de optimización multi-objetivo cumple la optimalidad de Pareto si al mejorar el resultado de uno de los objetivos, uno o más de los demás objetivos empeora su resultado. Además un punto se dice que es óptimo según la optimalidad de Pareto si no existe ningún otro punto que lo domine, es decir, que obtenga un mejor resultado alguno de los objetivos del problema. El conjunto de estos puntos se conoce como la frontera de Pareto. Existen fundamentalmente dos formas de abordar este tipo de problemas:

- Una de ellas es transformar la función objetivo, que recibe un vector de objetivos como parámetro, en una función que trabaje únicamente con un escalar de manera que puedan aplicarse alguno de los algoritmos mencionados anteriormente. Esto se puede realizar mediante 2 clases de métodos:
    - Basados en restricciones: estos métodos consisten en aplicar restricciones sobre la frontera de Pareto para obtener uno solo de los puntos óptimos del modelo. Esto puede hacerse de manera manual por un experto o de manera automática.
    - Basados en pesos: estos métodos asignan un peso a cada una de las funciones objetivo y en función del método elegido, se extraerá parte de la frontera de Pareto.
- La otra es utilizar algoritmos que identifican el conjunto de puntos para los que se alcanza un balance entre las diferentes funciones objetivo. Para ello pueden aplicarse versiones de los algoritmos poblacionales mencionados anteriormente que generan individuos que satisfagan estos requisitos. Una vez generados estos puntos se pueden presentar a un experto que diseñará el modelo final.

<u>Algoritmos para problemas indefinidos</u>

Este tipo de algoritmos de optimización trata de lidiar con problemas en los que la función objetivo o las restricciones no están del todo definidas ya sea por haber realizado aproximaciones sobre el modelo o fluctuaciones en los parámetros entre otros. De las posibles incertidumbres identificadas en el problema destacan 2 y cada una tiene sus técnicas particulares para crear modelos robustos:

- Incertidumbre basada en conjuntos:
  - Método minimax.
  - Teoría del salto de información.
- Utilizando aproximaciones probabilísticas. Éstas generalmente tratan de minimizar valores del modelo como:
  - La varianza.
  - El riesgo de inviabilidad del modelo.
  - Valor en riesgo que es el mejor resultado obtenible de la función objetivo para una cierta probabilidad.
  - Valor en riesgo condicional que es el valor esperado de la función objetivo para el cuartil 1-α de la distribución probabilística sobre el resultado del modelo.
  - Una combinación de los anteriores.

Optimización discreta

La optimización discreta se encarga de tratar problemas que no cuentan con variables continuas sino discretas. Es por ello que los problemas de optimización discreta cuentan con restricciones tales que las variables del problema deben elegirse sobre un conjunto discreto. Esto supone que algunos problemas de este tipo cuentan con infinitos espacios de trabajo y otros, conocidos como problemas de optimización combinatoria con un número finito de estos. En cualquiera de los dos casos no es posible computar todos los espacios posibles. Existen varios métodos dentro de este tipo de optimización y aquí únicamente se van a tratar dos de ellos.
En primer lugar están los programas enteros que son programas de optimización lineales cuyas restricciones se encuentran dentro del conjunto de números enteros. La estructura de los modelos de estos programas es similar a la utilizada en los modelos de optimización lineal en forma estándar, pero añadiendo la condición de pertenencia a los números enteros.
El segundo método es la programación dinámica. Para poder aplicar programación dinámica a un problema de optimización se deben dar dos condiciones:

- El problema debe tener una subestructura óptima. Esto significa que se puede construir una solución óptima a partir de las soluciones óptimas de sus subproblemas.
- El problema debe tener subproblemas superpuestos, es decir, que el problema puede dividirse en subproblemas que son reutilizados repetidamente o que, al ser resuelto mediante una función recursiva, esta se encuentre el mismo problema varias veces.

Para resolver el problema, la programación dinámica estructura el problema en forma de árbol de manera que cada nodo represente un subproblema del nodo anterior y a continuación puede tratar de resolver el problema de arriba abajo, es decir, desde el problema principal hacia los subproblemas o de abajo arriba, desde los subproblemas hasta el problema principal.

Ámbitos de uso más frecuentes

La técnica de optimización, sus principios y métodos se emplean en multitud de disciplinas como la Física, Ingeniería, Economía y Biología. En el sector industrial destacan la optimización de procesos

y la optimización de rutas. La optimización de procesos busca reducir o eliminar la pérdida de tiempo y recursos en el desarrollo de las actividades de empresas de manera que se haga un uso eficiente de los recursos y se obtenga el mayor beneficio. Los problemas de optimización de rutas consisten en encontrar el camino más corto en un grafo entre los diferentes nodos de la forma más eficiente posible. Esta aplicación de la optimización ha sido realmente útil en áreas como el transporte público y de mercancías, interconexión de redes, enrutado en internet entre otras.

Simulación

Para poder afrontar el diseño de nuevos productos (nuevas moléculas en el caso del sector farmacéutico) es necesario entender cómo se comportan los sistemas microscópicos, por lo que debe atender a las leyes de la mecánica cuántica.
Gracias a las observaciones sobre la dualidad onda-partícula a principios del siglo XX que explica el comportamiento de algunas partículas, como es el caso de los fotones (luz), se empezaron a entender fenómenos que la mecánica clásica no había sido capaz de resolver. Con el desarrollo de la ecuación de Schrödinger, pieza clave en la mecánica cuántica, se sentaron las bases de la mecánica cuántica que, a partir de entonces, se centró en dos líneas principales: (1) el desarrollo de modelos teóricos para la resolución aproximada de la ecuación de Schrödinger para sistemas multielectrónicos y (2) el desarrollo de métodos computacionales para la resolución de las ecuaciones provenientes de estos modelos, a fin de poder estudiar sistemas químicos reales.
Se produjo un gran avance con el desarrollo del método *ab-initio* Hartree-Fock (HF) que permite estudiar moléculas que sólo necesitan las constantes físicas fundamentales, como el tipo atómico, la masa y la posición inicial como parámetros sin necesidad de parámetros empíricos. Este método Hartree-Fock tenía como limitante que sólo podía aplicarse al estudio de sistemas pequeños (moléculas diatómicas). El problema fue resuelto con la aparición de los métodos semiempíricos, basados en el método Hartree-Fock con algunas aproximaciones.
En este punto, se desarrolla la teoría del funcional de la densidad (DFT, del inglés Density Field Theory), que supone que la energía de un sistema es un funcional de la densidad electrónica del sistema. Esta aproximación reduce drásticamente la complejidad de las ecuaciones a resolver, ya que la densidad electrónica es una función que depende sólo de tres coordenadas espaciales, siendo independiente del número de electrones. Los métodos basados en DFT proporcionan geometrías moleculares y propiedades termoquímicas con mayor exactitud que los métodos semiempíricos mencionados anteriormente y con un coste computacional mucho menor, lo que permite que se puedan aplicar al estudio de sistemas con un gran número de átomos, sistemas de interés real en la Química.
Estos métodos basados en DFT se pueden usar, entre otras cosas, para:

- calcular la energía de los orbitales moleculares,
- simulaciones de dinámicas moleculares,
- cálculo de potenciales de reacción.

T2.2: Identificación y selección de algoritmos de computación clásica relacionados con clasificación, optimización y simulación

En esta tarea, se procederá a la descripción del proceso propuesto para abordar, desde el punto de vista clásico, los tres problemas que componen la estructura básica del proyecto: clasificación, optimización y simulación. Se detallará tanto el planteamiento general del problema, los diferentes

conjuntos de datos tenidos en cuenta y el pre-procesado necesario para cada caso de uso, así como los modelos que se han contemplado, y que se validarán en tareas posteriores.

Clasificación

El primero de los casos de aplicación contemplados a lo largo del proyecto ALCATRAZ está asociado con métodos de clasificación. En particular, el problema de clasificación asociado se centrará en la generación de un sistema que permita clasificar ciberataques en función de su contenido.
Distributed denial-of-service (DDoS) es una forma de ataque hacia sistemas críticos como servidores, sitios web u otros recursos de red, con el objetivo de interrumpir temporal o permanentemente su conexión provocando la denegación de servicio para los usuarios.
En este caso, nos encontramos ante un problema de clasificación binaria, dado que se trata de un caso en el que clasificar un conjunto de datos en dos clases. Dentro del *machine learning*, la clasificación binaria se puede enmarcar como un problema de aprendizaje supervisado.

Planteamiento del problema

En primer lugar, se usará un conjunto de datos que contiene varios tipos de ataques DDoS (Netbios, Portmap y Syn), los cuales tienen lugar durante dos días diferentes. El conjunto de datos se dividirá en dos conjuntos: entrenamiento y pruebas o test. El conjunto de entrenamiento se utilizará para entrenar diferentes modelos para predecir el conjunto de pruebas/test clasificando conexiones DDOS o BENIGN en función de las variables independientes. La variable dependiente es una etiqueta de dos valores NETBIOS / PORTMAP (ataques DDoS) o BENIGN (conexión inofensiva) y las variables independientes son datos de red reales como la duración del flujo, los paquetes enviados por segundo, la longitud de los paquetes reenviados, etc.
En términos simples, el modelo observará los datos del día 1 y aprenderá qué datos de red están etiquetados como NETBIOS / PORTMAP o BENIGN e intentará predecir si las conexiones del día 2 son conexiones DDoS o conexiones benignas sin mirar sus etiquetas correctas. Las etiquetas predichas se comparan con las etiquetas reales para obtener un informe de clasificación que contenga precisión y tasas de error.

Conjunto de datos

Para el desarrollo de dicho sistema, es necesario disponer de un conjunto de datos adecuado para tal tarea, tal y como se ha indicado en el apartado previo dentro del propio planteamiento general del problema. Este conjunto de datos es accesible públicamente desde https://www.unb.ca/cic/datasets/ddos-2019.html.
Dado el gran volumen de información disponible para dicho conjunto, se han analizado las diferentes variables disponibles, seleccionando aquéllas que se han considerado de mayor relevancia, y son las 20 siguientes:

'Flow ID', 'Source IP', 'Source Port', 'Destination IP', 'Destination Port', 'Timestamp', 'Flow Duration', 'Total Fwd Packets', 'Total Backward Packets', 'Fwd IAT Mean', 'Fwd IAT Std', 'Fwd IAT Min', 'Fwd Packets/s', 'Bwd Packets/s', 'SYN Flag Count', 'RST Flag Count', 'PSH Flag Count', 'ACK Flag Count', 'URG Flag Count', 'Label'

La variable final de dicha lista será aquélla que contenga la información relativa a la variable objetivo. Dicho conjunto de datos consiste de un total de cinco datasets de interés, que se pasan a describir brevemente a continuación:

- Datos para el entrenamiento: aquellos recogidos a lo largo del día 12/1 del año de recogida, y que están a su vez divididos en dos:
  - DrDoS_NetBIOS.csv: con un tamaño de 4.094.986 entradas.
  - Syn.csv: con un tamaño de 1.582.681 entradas.
- Datos para el test: aquellos recogidos a lo largo del día 11/3 del año de recogida, y que están a su vez divididos en dos:
  - NetBIOS.csv: con un tamaño de 3.455.899 entradas.
  - Syn.csv: con un tamaño de 4.320.541 entradas.
  - Portmap.csv: con un tamaño de 191.694 entradas.

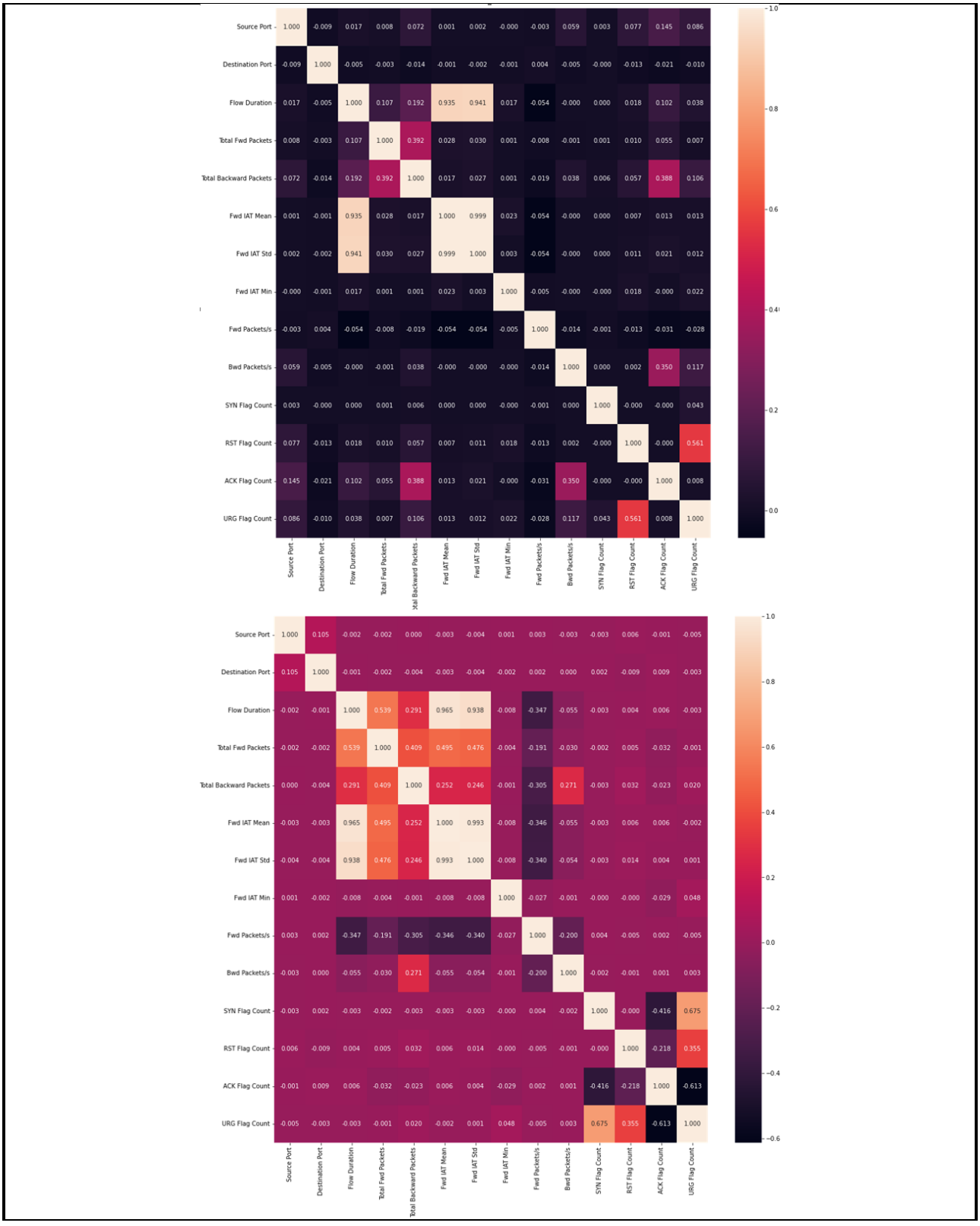
Tras la identificación del conjunto de datos a utilizar, se procederá a la limpieza de dicha información, de forma que se pueda adecuar a las exigencias de los modelos de aprendizaje supervisado a aplicar posteriormente a lo largo de este proyecto. Para ello, se han llevado a cabo diferentes pasos, como la eliminación de entradas que contengan valores no válidos en el modelado por ser valores infinitos o de entradas que contengan valores nulos. Del mismo modo, se han identificado aquellas variables que no proporcionan información variable a lo largo del dataset y son constantes, siendo la variable 'PSH Flag Count' aquélla que cumple dichas características y que, por tanto, se ha eliminado del estudio, pasando a tener conjuntos de 19 variables en total.

Tras realizar un análisis exploratorio de todos los datos disponibles, se llegan a las siguientes conclusiones asociadas:

- Datos de entrenamiento:
  - Los conjuntos de datos contienen 14 variables numéricas y 5 categóricas.
  - En los datos DrDoS_NetBIOS, 4.093.279 entradas se han etiquetado como ataques “DrDoS_NetBIOS”, mientras que 1.707 lo hacen con la etiqueta “BENIGN”.
  - En los datos Syn, 1.582.289 entradas se han etiquetado como ataques “Syn”, mientras que 392 lo hacen con la etiqueta “BENIGN”.
- Datos de test:
  - Los conjuntos de datos contienen 14 variables numéricas y 5 categóricas.
  - En los datos NetBIOS, 3.454.578 entradas se han etiquetado como ataques “NetBIOS”, mientras que 1.321 lo hacen con la etiqueta “BENIGN”.
  - En los datos Syn, 4.284.751 entradas se han etiquetado como ataques “Syn”, mientras que 35.790 lo hacen con la etiqueta “BENIGN”.
  - En los datos Portmap, 186.960 entradas se han etiquetado como ataques “Portmap”, mientras que 4.734 lo hacen con la etiqueta “BENIGN”.

De modo que sea posible unificar la información relativa a la variable objetivo, se transformará ésta en una variable binaria con posibles valores 0 y 1, donde los ataques serán etiquetados con el segundo, y los que sean considerados como benignos, con el primero. Así, dicha variable “Label” pasará a ser numérica con dichos dos posibles valores a lo largo de los diferentes conjuntos de datos tratados.

Como siguiente paso en el pre-procesado de dicha información, se han tomado los dos conjuntos de datos de entrenamiento, realizando para ambos de forma paralela un proceso de selección de variables, todo ello basado en la utilización de la conocida matriz de correlaciones. Se presentan ambas a continuación:

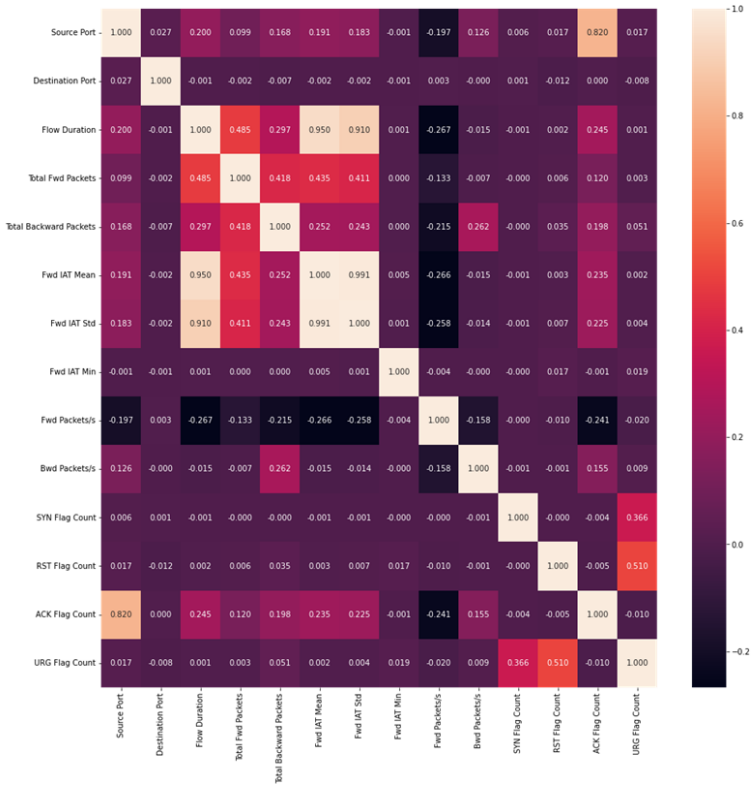

Source Port
Destination Port
Flow Duration
Total Fwd Packets
Total Backward Packets
Fwd IAT Mean
Fwd IAT Std
Fwd IAT Min
Fwd Packets/s
Bwd Packets/s
SYN Flag Count
RST Flag Count
ACK Flag Count
URG Flag Count

Atendiendo a ambas matrices de correlación, se pueden ver diferencias entre éstas, donde la segunda tiene mayor contraste en cuanto a valores extremos, si bien los valores más determinantes en ambas son comunes:

- Marcada correlación directa entre las variables "Flow Duration", "Fwd IAT Mean" y "Fwd IAT Std" (valores superiores a 0,9 en los diferentes pares de dichas tres variables para ambas fuentes de datos).

También es posible identificar correlaciones individuales que no son comunes entre sí, pero no se tratan de casos con valores tan extremos como los identificados en este punto. Así, se procederá a la eliminación de dos de dichas tres variables, de tal modo que se evite incluir en el modelado variables con información redundante, como sería este caso.

Sin embargo, dado que el estudio posterior se realizará con la unificación de ambas fuentes de datos como conjunto de entrenamiento, se procede a la generación de dicha matriz de correlaciones también de forma conjunta, y se presenta en la siguiente figura.



En este caso, se repite la relación a tres mencionada previamente de forma individual, si bien se identifican más relaciones relevantes, que se presentan también a continuación:

- Marcada correlación directa entre las variables "Flow Duration", "Fwd IAT Mean" y "Fwd IAT Std" (valores superiores a 0,9 en los diferentes pares de dichas tres variables).
- Marcada correlación directa entre las variables "Source Port" y "ACK Flag Count" (0,82).

De este modo, se han identificado dos grupos de variables cuya relación es suficientemente amplia, como para considerar acciones de selección de variables. Así, se podrán eliminar dos de las variables del primer grupo de éstas, y del mismo modo, eliminar otra del segundo par.

Dadas las características de los modelos habituales de clasificación, no se considerarán aquellas variables categóricas inicialmente en el modelado, y tras la eliminación de estas tres últimas variables

tras el análisis de correlaciones, se procederá a llevar a cabo el modelado posterior con un total de 11 variables, asociadas a 5.677.667 entradas diferentes.

Modelado

Una vez preparado el conjunto de datos a utilizar en este problema de clasificación, tal y como se ha descrito en los subapartados previos, se procederá con la fase propia del modelado. En ella, se seleccionarán diferentes algoritmos de aprendizaje automático supervisado de clasificación binaria que permitan llevar a cabo el proceso de categorización de nuevos ataques, basado en las variables asociadas, de tal modo que nuevos ataques desconocidos puedan ser clasificados única y exclusivamente con la información de sus variables independientes.

Regresión logística

El primero de los modelos considerados es el de la regresión logística. Se trata de uno de los métodos clásicos dentro de la temática, el cual se basa en la función logística para llevar a cabo el modelado de la variable objetivo (en este caso, si las características asociadas se corresponden o no a un ataque maligno).
Es un modelo que está inspirado en la conocida regresión lineal. En éste, la base es la generación de una función predictiva lineal, la cual busca proporcionar una puntuación a partir del conjunto de pesos de partida, y con la posterior combinación lineal de estos, junto a las propias variables del modelo.
Así, se trata de un modelo que permite generar un entrenamiento óptimo de los pesos, tal que sea posible llevar a cabo la transformación de la puntuación de forma directa en una probabilidad.

Máquinas de vector soporte

El segundo modelo considerado es el conocido como máquinas de vector soporte, denotado habitualmente como SVM por sus siglas en inglés (Support Vector Machine). Se trata de un modelo que se utiliza habitualmente en cualquier tipo de problemas de aprendizaje automático supervisado, si bien su concepción inicial surge para el tipo de problema que aquí nos encontramos, el de clasificación binaria.
La base de dicho método se fundamenta en la separación de las dos clases disponibles a través de la generación de un hiperplano (o un posible conjunto de estos), que permita separar ambas categorías de la forma más óptima posible.

K-vecinos

El siguiente modelo considerado se trata de uno de los métodos más sencillos a la hora de realizar un sistema de clasificación como el que aquí nos ocupa: se trata del método de los k-vecinos. Dicho modelo procede a clasificar un nuevo punto desconocido a través del conocimiento disponible de los k puntos más cercanos a éste. Para ello, es imprescindible disponer de una métrica que permita determinar cuáles son aquellos puntos más cercanos al nuevo a categorizar.

Árboles de decisión

Los árboles de decisión se tratan de un tipo de modelo ampliamente estudiado y aplicado para diferentes ramas del aprendizaje automático supervisado, incluyendo tanto casos de regresión como de clasificación.
Se tratan de algoritmos que se basan en la concatenación de diferentes decisiones lógicas, cuya resolución permite continuar a través de otras decisiones adicionales hasta llegar al punto final de resolución del modelo. El comienzo del recorrido se realiza desde el nodo inicial, procediendo a través de la decisión lógica inherente de cada nodo, y moviéndose a través de los ejes según la resolución de estos. Se pueden distinguir tres tipos de nodos: el nodo inicial previamente mencionado, los nodos internos, y los nodos hoja. Los internos presentarán nuevas decisiones lógicas para continuar con el proceso a través de distintos ejes, mientras que los nodos hoja serán los nodos finales donde se generará la salida proporcionada a la clasificación del individuo en estudio.
Dichas decisiones asociadas a los nodos inicial e internos tendrán como base las propias variables independientes del conjunto de datos en cuestión.

Random forest

Los árboles de decisión han servido de base para la generación de nuevas metodologías, y en concreto, la aquí tratada: los *random forest*. La base de dicho modelo es la agregación de varios árboles de decisión individuales, cuyas predicciones independientes son posteriormente agrupadas a través del método de agregación seleccionado. Dadas las características de dicho modelo, es evidente la importancia que tiene en su desarrollo el número de árboles que lo compone para su posterior agregación.

Modelos del descenso del gradiente

Se tratan de sistemas basados en dicha técnica, es decir, un algoritmo de optimización iterativa para la búsqueda de mínimos locales en funciones diferenciables. Los modelos del descenso del gradiente son formados por una serie de decisores combinados, donde cada uno de ellos estima el residuo que se ha obtenido por los previos mediante el método de optimización del descenso del gradiente. Así, se generará el resultado final a través de la suma de los resultados individuales obtenidos para cada uno de dichos decisores.
Dentro de este grupo, existen diferentes implementaciones que permiten su utilización en distintos tipos de problemas de aprendizaje automático supervisado, como el más conocido de todos, el modelo XGBoost (*Extreme Gradient Boosting*).

Optimización

El segundo caso abordado a lo largo de este proyecto estará asociado a un sistema de optimización. En particular, y al igual que ocurría en el caso previo de clasificación, se ha detallado a lo largo de las tareas correspondientes al paquete de trabajo previo PT1 que se trata de un problema de optimización de rutas, que permita realizar el reparto de una serie de productos de la mejor forma posible.
En este caso, la tipología del reparto estará basada en la necesidad de partir siempre del origen conocido donde recoger los productos, en el conocimiento de los detalles asociados a los productos a repartir, y en la existencia de una flota de diferentes características.

Planteamiento del problema

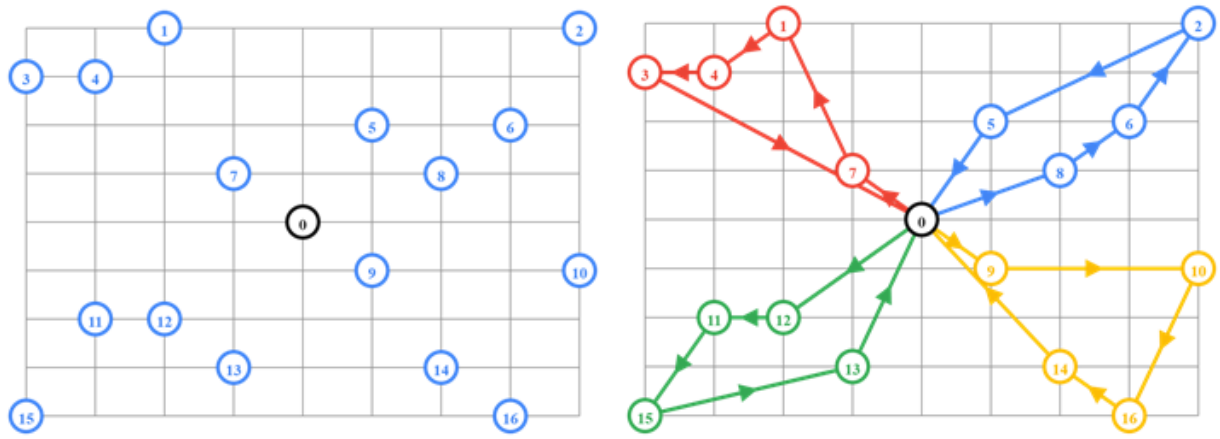
Se dispone de un conjunto de datos real, el cual será protegido por cuestiones de privacidad, y que será la base para el desarrollo de dicho problema de optimización de rutas. Para ello, se requerirá de un proceso exhaustivo de procesado de dichos datos, detallado a lo largo del apartado posterior, junto con una descripción de dicha información.
Tras la adecuación de dicha información, se procederá con la aplicación de algoritmos de optimización de rutas, que permitan generar los repartos óptimos minimizando el número de kilómetros totales a recorrer por toda la flota, y como consecuencia, el consumo asociado que permita minimizar tanto el gasto asociado como la contaminación adicional asociada a un reparto no óptimo.

Conjunto de datos

El conjunto de datos disponible para este apartado se corresponde con la información necesaria para la generación de un sistema de optimización de rutas. Dadas las características de dicha información, podremos distinguir varios subconjuntos de datos.
Por un lado, se dispone de la información relativa a la propia flota de reparto a utilizar. Se tratan de todos los vehículos que se podrán utilizar para realizar la distribución de los pedidos. Dicha flota está formada por 23 vehículos diferentes. Por otro lado, se dispone de la información asociada a los diferentes orígenes y destinos disponibles para el análisis del problema. La última variable de esta última fuente ("restricciones_vehiculo") se corresponde con información que indica qué posibles vehículos de la flota inicial pueden tener problemas para acceder a dichos destinos, suponiendo así restricciones adicionales en el problema de optimización de rutas.
Por último, también se ha proporcionado de forma complementaria y necesaria, información extraída de cada uno de los destinos con respecto a su geolocalización. Dicha información ha sido extraída previamente mediante la automatización con herramientas como *Open Street Map*, y con supervisión posterior de conocimiento experto para la corrección de posibles errores.
Por último, será necesario disponer de la información relativa a las distancias entre cada par de destinos, e incluyendo también los orígenes en dicho cálculo, dado que será parte indispensable en la base del modelado posterior. Para ello, se ha recurrido previamente a la misma herramienta abierta, *Open Street* Map a través de su API oficial, obteniendo un conjunto de datos que contempla cada par de puntos posibles.
Nótese que dichas distancias se han obtenido con metodologías específicas como la previamente mencionada, dado que no se trata de distancia geográfica sino por carretera, por lo que fórmulas como la distancia Haversine no es viable en esta situación, si bien su cálculo se ha realizado e incluido en la tabla, tal y como se puede apreciar.

Modelado

Dadas las características de este problema, tal y como se ha detallado a lo largo de esta tarea, se puede enmarcar el modelado a desarrollar dentro de un problema de optimización de rutas asociado a un problema del conocido tipo *Vehicle Routing Problem* (VRP). Del mismo modo que se ha explicado con anterioridad, el objetivo principal de estos problemas es el de buscar minimizar una variable objetivo, como en este caso el kilometraje total de toda la flota de reparto, y que, por consiguiente, se traduzca en reducciones del consumo que reporten ahorros tanto económico para la empresa como de contaminación al medioambiente. A modo ilustrativo, se presenta un ejemplo sencillo en la siguiente figura de dicho tipo de problema y su resolución.



Tal y como se puede apreciar, en la figura izquierda se presenta el origen (nodo 0), y los diferentes puntos a los que ha de entregarse diferentes pedidos, y a la derecha la solución de dicho problema. Se puede observar cómo se realiza en cuatro rutas diferentes, donde todas ellas tienen como punto inicial y final el origen.

Si bien este ejemplo se trata de un caso muy sencillo, dada la simplicidad de los datos iniciales, este tipo de problema sigue siendo un caso complejo que requiere de un alto coste computacional, por lo que es posible que no se puedan lograr los repartos óptimos globales. Cabe destacar además que, en este problema concreto, existen restricciones adicionales, tal y como se presenta en la Tabla 3, por lo que añade complejidad al sistema.

Por ello, en el modelado se considerará el uso de algoritmos evolutivos con base en métodos metaheurísticos en dos fases diferentes. Por un lado, se procederá con el cálculo de la solución de partida, la cual será utilizada como punto base para el resto del procedimiento. Tras esto, se irá refinando a través de dicha metodología metaheurística, de forma que sea posible obtener el óptimo global.

Dado que, tal y como se comenta previamente, estos sistemas pueden ser costosos computacionalmente, se ha procedido a limitar los tiempos de ejecución del modelo, de tal forma que se restrinja, por ejemplo, el número de iteraciones máximas que puede realizar el metaheurístico.

Dadas las características propias del problema, se ha seleccionado para su desarrollo la librería de optimización desarrollada por Google para tal fin, bajo el nombre or-tools, y que es accesible a través del lenguaje de programación Python.

Esta librería permite acceder a diferentes herramientas para la resolución de problemas de optimización de diferentes tipos, entre los que se incluyen los VRP que se contemplan en este proyecto. No solo eso, sino que también permite añadir restricciones particulares del problema, como las que se han indicado previamente asociadas a la accesibilidad de la flota a según qué destinos.

Dentro de los algoritmos de búsqueda de solución inicial, se considerarán diferentes tipos de metodologías: métodos por arco, *savings*, *sweep* o métodos de inserción de nodos. Dichos métodos se describirán, analizarán en profundidad y compararán a lo largo de este proyecto, y se detallarán en las tareas posteriores.

Lo mismo ocurre con el caso de los algoritmos metaheurísticos. Se llevará a cabo un análisis similar al caso de búsqueda de solución inicial, considerando opciones como *greedy descent*, *simulated annealing*, *tabu search* o *guided local search*.

Simulación

Planteamiento del problema

En el caso del uso de simulación en el diseño de nuevas moléculas dentro del sector farmacéutico, se plantean diferentes ejemplos en los que puede ser interesante el uso de computación cuántica. Analizamos cada uno de ellos, sus requisitos (datos de entrada) y cómo se resuelven en la actualidad mediante técnicas de computación clásicas.

1. **Optimización geométrica de moléculas simples**

La geometría molecular es la disposición tridimensional de los átomos de una molécula. La predicción de la posición más estable de los átomos de una molécula es un problema de optimización en el cual se minimiza la energía de la molécula respecto a las posiciones de los núcleos de los átomos que la componen.
La geometría molecular se puede deducir aplicando la Teoría de Repulsión de los Pares de Electrones de la Capa de Valencia (RPECV) teniendo en cuenta que todos los pares de electrones se repelen entre sí y que adoptan la configuración que minimice estas fuerzas de carácter repulsivo en torno al átomo central.
En ALCATRAZ, planteamos empezar modelando moléculas pequeñas (entre 4 y 12 átomos) con diferentes geometrías bien conocidas (CH4, NH3, Benceno, CP*, etc.).
En todos los casos, se usarán como punto de partida geometrías ligeramente distorsionadas para poder confirmar que el cálculo cuántico lleva a cabo el proceso de optimización geométrica de forma adecuada.

2. **Cálculo de orbitales en compuestos sencillos**

Este cálculo está enfocado principalmente en las energías del HOMO (del inglés, *Highest Occupied Molecular Orbital*) y del LUMO (del inglés, *Lowest Occupied Molecular Orbital*), comúnmente llamados orbitales frontera, para así poder obtener parámetros como la energía de excitación, que es indicativa de la reactividad de una molécula.
Existen diferentes programas de cálculo que se pueden utilizar para calcular los orbitales HOMO y LUMO. En sus inicios, la entrada de datos en estos programas se realizaba mediante un archivo de texto en el que era preciso introducir manualmente las coordenadas de los átomos en la molécula y el programa generaba otro archivo de texto que incluía los resultados del cálculo. Actualmente, programas como HyperChem, utilizan una interfaz gráfica que facilita la entrada de datos y la visualización de los resultados.
Para esta aproximación, se plantea el uso de moléculas con enlaces múltiples, como el vinilo (tres átomos) o el benceno (seis átomos).

3. **Simulaciones de Dinámicas Moleculares**

Se plantea analizar algún proceso simple con un número bajo de átomos, como por ejemplo, la formación de puentes de hidrógeno intermoleculares en un sistema con pocas moléculas de agua o el intercambio protónico en un sistema ácido/base.
La MD es una técnica de simulación por ordenador que permite a átomos y moléculas interaccionar durante un período de tiempo con una visualización del movimiento de las partículas.
La función de energía potencial son las fuerzas de atracción y repulsión que experimentan los átomos entre sí en los enlaces químicos, interacciones electrostáticas, fuerzas de van der Waals, etc. de las

moléculas. Se trata de una función de las coordenadas de las partículas. Estos cálculos se realizan a menudo a través de la mecánica cuántica.
Para llevar a cabo todas estas simulaciones, existe una gran variedad de software. De cara a proceder a la mejor elección para este proyecto, las únicas restricciones claras, de momento, son la posibilidad de realizar los cálculos propuestos, en especial las dinámicas moleculares, ya que optimizaciones geométricas y cálculo de orbitales las realizan prácticamente todos los softwares.
La idea es realizar todos los cálculos con un mismo paquete de software. Entre las principales posibilidades, bien por haber una documentación más extensa o bien por tener experiencia previa, nos encontramos con los siguientes programas: CPMD, cp2k, NWChem, Q-espresso y los módulos de Python Psi4 y PyQuante. Los módulos de Python cuentan, como principal ventaja, con una mayor facilidad a la hora de instalarlo e incluso de modificarlo si así se necesitase. Por el contrario, los programas propuestos cuentan con mayor oferta de opciones y han sido ampliamente usados por la comunidad científica. Otras opciones muy conocidas y aceptadas también por la comunidad, como por ejemplo Gaussian o ADF, han sido descartadas por tratarse de software comercial y considerar que sobrepasarían las necesidades del proyecto.

Conjunto de datos

Para llevar a cabo estas simulaciones, se proporcionarán como datos de entrada las geometrías de las moléculas pertinentes en formato .xyz (véase siguiente figura), al considerarlo uno de los formatos más ampliamente aceptados por los diversos softwares, tanto de simulación como de visualización. Estas geometrías iniciales, como la que se muestra en el ejemplo, serán suministradas por ProtoQSAR, como datos de partida para las modelizaciones moleculares.

```
9
Esta línea es simplemente para incluir información pertinente
H          -2.0801425360     0.4329727646     0.0722817289
C          -1.2129704155    -0.2295285634    -0.0097156258
H          -1.2655910941    -0.9539857247     0.8097953440
C           0.0849758188     0.5590385475     0.0510545434
O           1.2322305822    -0.2731895077    -0.1276123902
H           0.1506137362     1.1200249874     0.9943015309
H           1.2473876659    -0.8998737590     0.6150681570
H           0.1316093068     1.2841805400    -0.7645223601
H          -1.2737541560    -0.7748626513    -0.9540587845
```

Figura 1 - Archivo en formato xyz para la molécula de etanol (C2H6O).

Modelado

El comportamiento y propiedades de las moléculas vienen determinados por las interacciones de sus componentes. Estas interacciones de partículas atómicas y subatómicas obedecen a los principios de la mecánica cuántica. En ésta, el operador **Hamiltoniano** describe la energía total del sistema. Según la ecuación de Schrödinger, el Hamiltoniano (H) describe la evolución temporal de la función de onda del sistema cuántico en base al estado en tiempo cero. Debido a su complejidad, sólo es posible resolverla de forma exacta para un sistema monoelectrónico (átomo de hidrógeno). Es por eso por lo que existen modelos teóricos que permiten resolver de manera aproximada la ecuación de Schrödinger para sistemas multielectrónicos.

El estudio de moléculas pequeñas se hizo posible mediante el **método de Hartree-Fock**, y el de sistemas mayores mediante **métodos semiempíricos** basados en Hartree-Fock.
La teoría del funcional de la densidad (**DFT**, por sus siglas en inglés) ofrece una alternativa valiosa que es más eficiente que los métodos Hartree-Fock y, sin embargo, es bastante precisa. Esta teoría se basa en el teorema de Hohenberg-Kohn, que muestra que es posible usar solo la densidad electrónica para calcular la energía del estado fundamental E0. De ahí que la complejidad del problema disminuya drásticamente, ya que la densidad electrónica es función de solo tres coordenadas, y es independiente del número de electrones.
El teorema de Hohenberg-Kohn también demuestra que se cumple un principio variacional para el funcional de energía: para cualquier densidad, la energía dada por el funcional de energía correspondiente nunca es más pequeña que la energía del estado fundamental, y la energía del estado fundamental se obtiene sólo usando la densidad electrónica exacta del estado fundamental.

T2.3: Diseño e implementación de algoritmos de computación clásica relacionados con clasificación, optimización y simulación

Esta anualidad, se ha iniciado con el desarrollo parcial de la tarea T2.3, tal y como refleja el cronograma inicial del proyecto. En este caso, se ha comenzado con el diseño e implementación de cada uno de los tres problemas que componen el esquema del proyecto: clasificación, optimización y simulación.
En particular, se han tomado los modelos seleccionados, tal y como se ha detallado en la tarea previa, llevando a cabo las implementaciones correspondientes, y procediendo al enlace con las técnicas de pre-procesado inicial que también han sido destacadas en dicha tarea.
Por un lado, para el caso asociado al problema de clasificación planteado, la detección de ataques DDoS basados en el conjunto de datos recabados, se ha procedido con el diseño e implementación inicial de diferentes modelos de los ya planteados en la documentación asociada a la tarea previa, "T2.2: Identificación y selección de algoritmos de computación clásica relacionados con clasificación, optimización y simulación". En particular, se han considerado, por el momento, los modelos de máquinas de vector soporte, árboles de decisión, *Random Forest*, y modelos del descenso del gradiente como el XGBoost. Todos ellos, han sido implementados a través del lenguaje de programación Python, y librerías específicas para tal fin, como *sklearn* o *xgboost*.
Además, se han complementado todos ellos con diferentes opciones para sus propios hiperparámetros, como el kernel en el caso de las máquinas de vector soporte, la profundidad de los árboles de decisión, el número de árboles que componen el *Random Forest* o el booster que define el modelo XGBoost. Así, el objetivo será definir un proceso que compruebe diferentes combinaciones de hiperparámetros de cada uno de ellos, para pasar a su comparación experimental a lo largo de la tarea posterior, incluida de forma total en la anualidad del 2022.
Con respecto al segundo caso de uso planteado, centrado en un problema de optimización de rutas, se ha comenzado del mismo modo con el diseño y desarrollo de los diferentes modelados posibles para tal problemática, de nuevo, basado en el conjunto de datos ya seleccionado inicialmente y previamente explicado. Para ello, el diseño se ha basado en el uso de algoritmos evolutivos y que a su vez, están basados en métodos metaheurísticos a través de dos fases distintas: la generación de la solución inicial y el refinamiento mediante la metodología metaheurística correspondiente. Al igual que en el caso de uso previo, se ha procedido a través del lenguaje de programación Python y, en este caso, entre las librerías más relevantes para su desarrollo, se ha considerado la conocida como *or-tools*, que ha sido desarrollada por Google para problemas de dicha temática.

Finalmente, con respecto al tercer caso de uso, relativo a la simulación, por el momento se ha comenzado con un diseño inicial de los que se considerarán modelos asociados, si bien se encuentra en una fase menos desarrollada por el momento que los dos casos previos, y que se cerrará a lo largo del resto de desarrollo de la tarea actual correspondiente a la anualidad del 2022.
De este modo, se sientan las bases iniciales para que, en la anualidad próxima, correspondiente al año 2022, se continúe y cierre la implementación de dichos tres problemas con sus respectivos algoritmos seleccionados, y considerando, si así se decidiera, la posible inclusión de búsquedas intensivas de parámetros de estos (*parameter tuning*) o la selección de variables asociada de forma más detallada.
Esto irá estrechamente ligado a la validación de dichos algoritmos que se desarrollará íntegramente a lo largo de la anualidad 2022, y que se corresponde con la tarea posterior T2.4.
A modo ilustrativo, se presenta parte del código fuente relativo al problema asociado a la clasificación de ataques DDoS, incluyendo los pasos previos de preparación de los datos, y la de modelado desarrollada hasta el momento, incluyendo los cuatro modelos previamente mencionados, junto a parametrizaciones a rastrear, y que se evaluarán a lo largo de la próxima anualidad.

```
import pandas
import numpy
from sklearn.model_selection import GridSearchCV

# Put column names into a list
header_names = list(pandas.read_csv('ddos_datasets/03-11/Portmap.csv', nrows=0))

# Remove leading and trailing whitespaces
for i in range(0, len(header_names)):
   header_names[i] = header_names[i].strip()

# List of columns to use
use_columns = ["Flow ID", "Source IP", "Source Port", "Destination IP", "Destination Port", "Timestamp", "Flow Duration", "Total Fwd Packets", "Total Backward Packets", "Fwd IAT Mean", "Fwd IAT Std", "Fwd IAT Min", "Fwd Packets/s", "Bwd Packets/s","SYN Flag Count", "RST Flag Count", "PSH Flag Count", "ACK Flag Count", "URG Flag Count", "Label"]

# USN datasets
# Use header_names extracted previously
dataframe_train_net = pandas.read_csv('ddos_datasets/01-12/DrDoS_NetBIOS.csv', names = header_names, skiprows=1, usecols=use_columns)
dataframe_train_syn = pandas.read_csv('ddos_datasets/01-12/Syn.csv', names = header_names, skiprows=1, usecols=use_columns)
dataframe_test_net = pandas.read_csv('ddos_datasets/03-11/NetBIOS.csv', names = header_names, skiprows=1, usecols=use_columns)
dataframe_test_syn = pandas.read_csv('ddos_datasets/03-11/Syn.csv', names = header_names, skiprows=1, usecols=use_columns)
dataframe_portmap = pandas.read_csv('ddos_datasets/03-11/Portmap.csv', names = header_names, skiprows=1, usecols=use_columns)

dataframe_train_net = dataframe_train_net.replace([numpy.inf, -numpy.inf], numpy.nan)
dataframe_train_syn = dataframe_train_syn.replace([numpy.inf, -numpy.inf], numpy.nan)
dataframe_test_syn = dataframe_test_syn.replace([numpy.inf, -numpy.inf], numpy.nan)
dataframe_test_net = dataframe_test_net.replace([numpy.inf, -numpy.inf], numpy.nan)
dataframe_portmap = dataframe_portmap.replace([numpy.inf, -numpy.inf], numpy.nan)

dataframe_train_net = dataframe_train_net.dropna()
```

```
dataframe_train_syn = dataframe_train_syn.dropna()
dataframe_test_net = dataframe_test_net.dropna()
dataframe_test_syn = dataframe_test_syn.dropna()
dataframe_portmap = dataframe_portmap.dropna()

dataframe_train_net = dataframe_train_net.loc[:, (dataframe_train_net != 0).any(axis=0)]
dataframe_train_syn = dataframe_train_syn.loc[:, (dataframe_train_syn != 0).any(axis=0)]

dataframe_test_net = dataframe_test_net.loc[:, (dataframe_test_net != 0).any(axis=0)]
dataframe_test_syn = dataframe_test_syn.loc[:, (dataframe_test_syn != 0).any(axis=0)]
dataframe_portmap = dataframe_portmap.loc[:, (dataframe_portmap != 0).any(axis=0)]

# Change of label
dataframe_train_net['Label'] = [1.0 if x == "DrDoS_NetBIOS" else 0.0 for x in dataframe_train_net['Label']]
dataframe_train_syn['Label'] = [0.0 if x == "Syn" else 1.0 for x in dataframe_train_syn['Label']]
dataframe_test_net['Label'] = [1.0 if x == "NetBIOS" else 0.0 for x in dataframe_test_net['Label']]
dataframe_test_syn['Label'] = [0.0 if x == "Syn" else 1.0 for x in dataframe_test_syn['Label']]
dataframe_portmap['Label'] = [1.0 if x == "Portmap" else 0.0 for x in dataframe_portmap['Label']]

# Correlation matrices
corr_matrx_net = dataframe_train_net.drop(columns = "Label").corr()
corr_matrx_syn = dataframe_train_syn.drop(columns = "Label").corr()
corr_matrx = dataframe_train_net.append(dataframe_train_syn).drop(columns = "Label").corr()

# Include a list of required features within dataframe_train[]
required_features = ["Flow Duration", "Total Fwd Packets", "Total Backward Packets", "Fwd IAT Mean", "Fwd IAT Std",
"Fwd IAT Min", "Fwd Packets/s", "Bwd Packets/s", "RST Flag Count", "ACK Flag Count", "URG Flag Count"]
dataframe_features = dataframe_train_net.loc[:,required_features]

# Create X_train and Y_train variables
Y_train_1 = dataframe_train_net['Label'].values
X_train_1 = dataframe_features.values

############
### SVM ###
############

from sklearn import svm

# Grid search
parameters = [
 {'C': [0.01, 1.0, 100.0],
  'kernel': ['linear']},
 {'C': [0.01, 1.0, 100.0],
  'gamma': [0.01, 1.0, 100.0],
  'degree': [1, 3, 5],
  'kernel': ['poly']},
 {'C': [0.01, 1.0, 100.0],
  'gamma': [0.01, 1.0, 100.0],
  'kernel': ['rbf']},
 {'C': [0.01, 1.0, 100.0],
  'gamma': [0.01, 1.0, 100.0],
  'kernel': ['sigmoid']}
 ]

svm_grid_search = GridSearchCV(svm.SVC(random_state = 1), parameters, verbose = 10)
svm_grid_search.fit(X_train_1, Y_train_1)
```

```
if svm_grid_search.best_params_["kernel"] == 'linear':
    svm_clf = svm.SVC(C = svm_grid_search.best_params_["C"],
                      kernel = 'linear', random_state = 1)
elif svm_grid_search.best_params_["kernel"] == 'poly':
    svm_clf = svm.SVC(C = svm_grid_search.best_params_["C"],
                      gamma = svm_grid_search.best_params_["gamma"],
                      degree = svm_grid_search.best_params_["degree"],
                      kernel = 'poly', random_state = 1)
elif svm_grid_search.best_params_["kernel"] == 'rbf':
    svm_clf = svm.SVC(C = svm_grid_search.best_params_["C"],
                      gamma = svm_grid_search.best_params_["gamma"],
                      kernel = 'rbf', random_state = 1)
elif svm_grid_search.best_params_["kernel"] == 'sigmoid':
    svm_clf = svm.SVC(C = svm_grid_search.best_params_["C"],
                      gamma = svm_grid_search.best_params_["gamma"],
                      kernel = 'sigmoid', random_state = 1)
    svm_clf = svm.SVC(C = svm_grid_search.best_params_["C"],
                      gamma = svm_grid_search.best_params_["gamma"],
                      kernel = 'sigmoid', random_state = 1)

##########
### DT ###
##########

from sklearn import tree

# Grid search
parameters = {'criterion': ['gini', 'entropy'],
              'splitter': ['best', 'random'],
              'max_depth': [None, 1, 2, 4, 8, 16, 32, 64],
              'min_samples_split': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
              'min_samples_leaf': [0.1, 0.2, 0.3, 0.4, 0.5],
              'max_features': [None, "auto", "log2"]}
dt_grid_search = GridSearchCV(tree.DecisionTreeClassifier(random_state = 1), parameters, verbose = 10)
dt_grid_search.fit(X_train_1, Y_train_1)

dt_clf = tree.DecisionTreeClassifier(criterion = dt_grid_search.best_params_["criterion"],
                                     max_depth = dt_grid_search.best_params_["max_depth"],
                                     min_samples_split = dt_grid_search.best_params_["min_samples_split"],
                                     min_samples_leaf = dt_grid_search.best_params_["min_samples_leaf"],
                                     max_features = dt_grid_search.best_params_["max_features"],
                                     random_state = 1)

######################
### Random Forest ###
######################

from sklearn.ensemble import RandomForestClassifier

# Grid search
parameters = {'n_estimators': [10, 50, 100],
              'criterion': ['gini', 'entropy'],
              'max_depth': [None, 1, 4, 16, 32],
              'min_samples_split': [0.1, 0.5, 1.0],
              'min_samples_leaf': [0.1, 0.3, 0.5],
              'max_features': [None, "auto", "log2"]}
```

```
rf_grid_search = GridSearchCV(RandomForestClassifier(random_state = 1), parameters, verbose = 10)
rf_grid_search.fit(X_train_1, Y_train_1)

rf_clf = RandomForestClassifier(n_estimators = rf_grid_search.best_params_["n_estimators"],
                        criterion = rf_grid_search.best_params_["criterion"],
                        max_depth = rf_grid_search.best_params_["max_depth"],
                        min_samples_split = rf_grid_search.best_params_["min_samples_split"],
                        min_samples_leaf = rf_grid_search.best_params_["min_samples_leaf"],
                        max_features = rf_grid_search.best_params_["max_features"],
                        random_state = 1)

####################
### XGBoost Tree ###
####################

import xgboost as xgb

# Grid search
parameters = {"learning_rate": [0, 0.5, 1.0],
          "gamma" : [0, 1, 5],
          "max_depth": [10, 100, 200],
          "min_child_weight": [1, 10,  20],
          "subsample": [0.1, 0.5, 1.0],
          "reg_lambda": [1, 3, 5],
          "reg_alpha": [0.01, 0.5, 1.0]}
xgbt_grid_search = GridSearchCV(xgb.XGBClassifier(booster = 'gbtree', random_state = 1,
                                eval_metric = 'logloss', use_label_encoder = False),
                        parameters, verbose = 10)
xgbt_grid_search.fit(X_train_1, Y_train_1)

xgbt_clf = xgb.XGBClassifier(learning_rate = xgbt_grid_search.best_params_["learning_rate"],
                     gamma = xgbt_grid_search.best_params_["gamma"],
                     max_depth = xgbt_grid_search.best_params_["max_depth"],
                     min_child_weight = xgbt_grid_search.best_params_["min_child_weight"],
                     subsample = xgbt_grid_search.best_params_["subsample"],
                     reg_lambda = xgbt_grid_search.best_params_["reg_lambda"],
                     reg_alpha = xgbt_grid_search.best_params_["reg_alpha"],
                     booster = 'gbtree', random_state = 1, eval_metric = 'logloss',
                     use_label_encoder = False)

######################
### XGBoost Linear ###
######################

import xgboost as xgb

# Grid search
parameters = {"reg_lambda": [1, 3, 5],
          "reg_alpha": [0.01, 0.5, 1.0]}
xgbl_grid_search = GridSearchCV(xgb.XGBClassifier(booster = 'gblinear', random_state = 1,
                                eval_metric='logloss', use_label_encoder = False),
                        parameters, verbose = 10)
xgbl_grid_search.fit(X_train_1, Y_train_1)

xgbl_clf = xgb.XGBClassifier(reg_lambda = xgbl_grid_search.best_params_["reg_lambda"],
                     reg_alpha = xgbl_grid_search.best_params_["reg_alpha"],
                     booster = 'gblinear', random_state = 1,
```

```
                eval_metric = 'logloss', use_label_encoder = False)
```

PT3: Análisis, diseño y desarrollo de algoritmos basados en computación cuántica

Este paquete de trabajo se organiza en torno a cuatro tareas, de las cuales una ha sido iniciada y finalizada en su totalidad en esta anualidad, y una segunda ha sido iniciada con previsión de finalización en el año 2022. Las dos restantes, se iniciarán y finalizarán a lo largo de la próxima anualidad.

- T3.1: Análisis del estado de la técnica en algoritmos de computación cuántica relacionados con clasificación, optimización y simulación
- T3.2: Identificación y selección de algoritmos de computación cuántica relacionados con clasificación, optimización y simulación (iniciada, pero no finalizada)

A continuación, se describen las acciones desarrolladas en la tarea correspondiente por completo a esta anualidad y los principales resultados alcanzados. En el caso de la tarea iniciada, pero no finalizada, se dará una breve descripción de los desarrollos realizados hasta el momento y que se presentarán de forma finalizada en la próxima anualidad.

T3.1: Análisis del estado de la técnica en algoritmos de computación cuántica relacionados con clasificación, optimización y simulación

Esta tarea presenta una visión de alto nivel del estado de la técnica dentro del campo de la computación cuántica aplicado a tres dominios:

- **Simulación** dentro del dominio de la química para, por ejemplo, optimización de geometrías moleculares.
- **Optimización** de modelos para la resolución de problemas difíciles (i.e. NP-Hard) cuyos tiempos de ejecución en procesadores clásicos es desmesurado.
- **Clasificación** predictiva de etiquetas a partir de un conjunto de datos de entrada. De manera general, esto incluye el campo de Quantum Machine Learning (QML).

Estos dominios se asientan sobre el concepto de los circuitos variacionales, circuitos cuánticos compuestos de una secuencia de puertas cuánticas unitarias. La propiedad unitaria tiene especial importancia, ya que permite asegurar que la magnitud (norma) del vector de estado cuántico no cambia, es decir, que el vector sigue representando una distribución de probabilidad a lo largo de la secuencia de transformaciones. Los parámetros de estas puertas pueden a su vez ser sujetos a un proceso de optimización (entrenamiento) para minimizar una función de coste arbitraria que permita resolver un problema particular. En la sección de optimización se incluye más contexto a este respecto.

Además, en las siguientes secciones, dentro del contenido de esta tarea, se hace referencia a dos tipos de ordenadores cuánticos. Para mayor claridad, se describen en esta lista:

- Noisy Intermediate-Scale Quantum (NISQ): Estos son los ordenadores que están actualmente disponibles a la comunidad en general. Se caracterizan por estar limitados en el número de qubits disponibles; esta limitación es aún más pronunciada cuando se implementan mecanismos de corrección de errores. IBM Quantum, un servicio que ofrece acceso a un conjunto de ordenadores cuánticos, es uno de los representantes más relevantes de esta categoría.
- Fault-Tolerant Quantum Computers (FTQC): Ordenadores cuánticos con un alto número de qubits y capacidades de corrección de errores avanzadas. Su diseño y construcción es uno de los focos actuales de la comunidad científica, por lo que se tiene esperanza en que estén disponibles en un futuro razonablemente cercano. Permitirán implementar procesos con ganancias reales de rendimiento de varios órdenes de magnitud sobre los procesadores clásicos.

Este panorama se representa en la siguiente imagen:

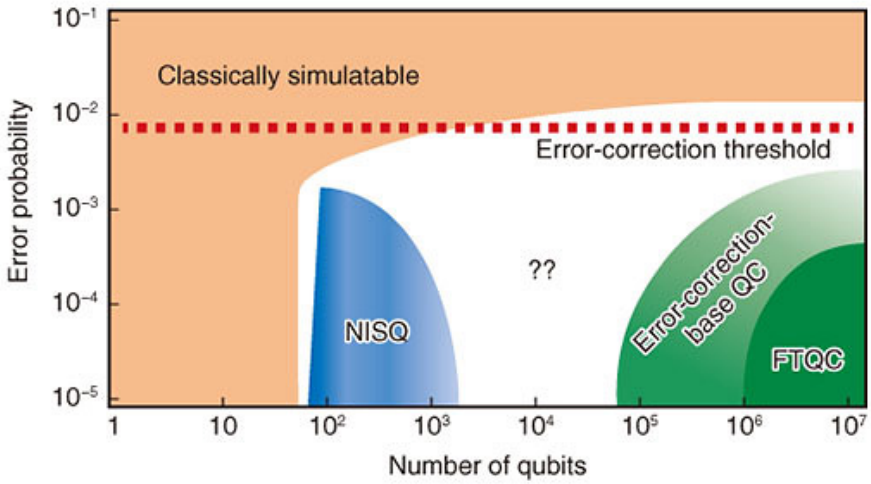


Figura 2 - Panorama de dispositivos de computación cuántica. Imagen extraída de (William John Munro et al., 2021).

La información contenida en este documento está clasificada bajo el modelo de computación cuántica basado en circuitos cuánticos. Se podría argumentar, dada la prevalencia de los recursos relacionados (e.g. frameworks, referencias académicas) que este es el modelo más popular. Aun así, se debe mencionar que existen otros modelos alternativos de computación cuántica, por ejemplo, la computación cuántica adiabática (Farhi et al., 2000).

Finalmente, se debe destacar que la computación cuántica es un campo con mucho potencial, pero que se encuentra en una etapa muy temprana. Las referencias académicas demuestran sobradamente de manera formal las posibilidades de este nuevo paradigma. Sin embargo, hay una escasez de ordenadores cuánticos reales capaces de implementar algoritmos de producción que demuestren mejor rendimiento que las alternativas clásicas. De hecho, este punto de la "supremacía cuántica" es una cuestión razonablemente polémica, que aún está en proceso de ser plenamente despejada de manera contrastada.

Simulación

El comportamiento y propiedades de las moléculas viene determinado por las interacciones de sus componentes. Estas interacciones de partículas atómicas y subatómicas obedecen a los principios de la mecánica cuántica, por lo que resulta intuitivo que los ordenadores cuánticos pueden ofrecer posibilidades interesantes para su modelización y simulación. El dominio de la química es uno de los casos de uso de aplicación de la computación cuántica con potencial significativo (Cao et al., 2019).

Un operador Hamiltoniano (o Hamiltoniano simplemente) describe la energía total de un sistema cuántico. Este tipo de entidades son actualmente uno de los actores principales que se utilizan en las técnicas de simulación de sistemas cuánticos, como es el caso en química. Según la ecuación de Schrödinger, el Hamiltoniano (H) describe la evolución temporal de la función de onda (de manera gruesa, el statevector) del sistema cuántico en base al estado en tiempo cero:

$$|\psi(t)\rangle = e^{-iHt}|\psi(0)\rangle$$

Implementar esta ecuación en un sistema clásico crece rápidamente en complejidad con el tamaño de las partículas, hasta el punto en el que el problema es inabarcable. Esta es una de las mayores limitaciones de los ordenadores clásicos en lo que respecta a simulación química.

Uno de los focos actuales de la comunidad científica es el diseño de técnicas para la explotación de dispositivos NISQ; esta necesidad se origina por la indisponibilidad de dispositivos FTQC, que se espera que aparezcan en un futuro. Esto ha dado lugar a un paradigma conocido como algoritmos cuánticos variacionales (variational quantum algorithms).

Los variational quantum algorithms adoptan una aproximación híbrida que combina fases de procesamiento clásico con etapas de procesamiento cuántico; de esta manera se maximiza la aportación de los dispositivos cuánticos dentro de sus capacidades realistas. De manera general, los parámetros de entrada al circuito cuántico se optimizan de manera iterativa según métodos clásicos para minimizar el coste determinado por el Hamiltoniano, es decir, para llegar al estado fundamental (i.e. ground state) del sistema.

A continuación se describe en mayor detalle el algoritmo VQE. El razonamiento es que VQE se identifica como el actor de mayor relevancia en la simulación química con computación cuántica, así como la inspiración directa de otras técnicas similares. Una aproximación basada en VQE resulta suficiente para obtener una vista amplia de este dominio.

VQE

Atendiendo al volumen de referencias y recursos relacionados, es razonable argumentar que Variational Quantum Eigensolver (VQE) (Peruzzo et al., 2014) es el representante más relevante dentro del dominio de la computación cuántica aplicada a la simulación química. VQE, como principal representante de los variational quantum algorithms, proporciona una solución híbrida para la optimización de Hamiltonianos moleculares. Este proceso de optimización tiene como objetivo último obtener el ground state del sistema descrito por el Hamiltoniano.

La siguiente imagen proporciona una vista de alto nivel del algoritmo VQE que resulta útil para entender su flujo y funcionamiento. Además, se describe a continuación de manera general el flujo de VQE.

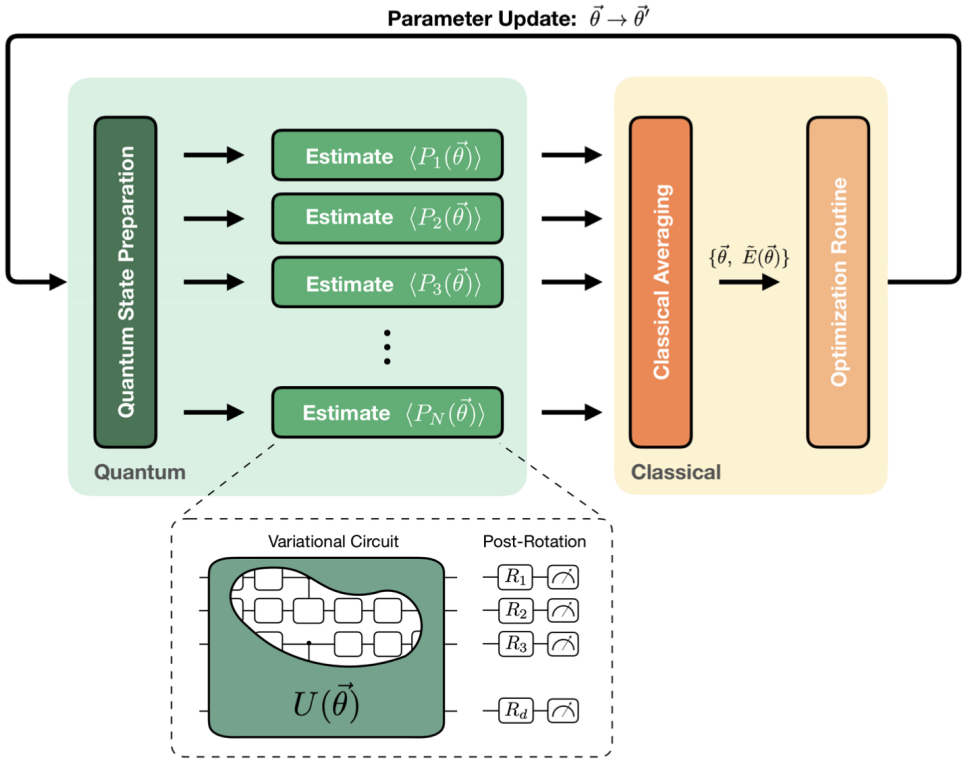


Figura 3 - Esquema del algoritmo VQE. Imagen extraída de (Cao et al., 2019).

1. El primer paso consiste en definir el Hamiltoniano que describe la estructura de la molécula bajo estudio. Este Hamiltoniano toma la forma de una combinación lineal de puertas cuánticas de tipo Pauli. El número de qubits viene a su vez determinado por el número de electrones de la molécula en cuestión, concretamente, el número de orbitales posibles. El proceso de construcción del Hamiltoniano se basa en el método de Hartree–Fock, un método iterativo para solucionar de manera numérica la ecuación de Schrödinger independiente en tiempo.
2. Seguidamente se debe construir el circuito variacional (Variational Circuit en el diagrama) que será el objetivo del proceso de optimización. El circuito actúa como entrada, es decir, define el estado bajo prueba de la molécula. En otras palabras, el circuito variacional permite definir una superposición de los estados válidos de la molécula, donde los qubit se utilizan para determinar la presencia de un electrón en un orbital. Existen diferentes estrategias para el diseño del circuito variacional, incluyendo aproximaciones por "fuerza bruta" en las que se codifican todas las rotaciones existentes, así como métodos aproximados que producen circuitos variacionales de menor profundidad (es decir, que requieren menos recursos computacionales).
3. La función de coste a optimizar es la esperanza del Hamiltoniano para el estado determinado por el circuito variacional. Esta función de coste se optimiza mediante técnicas clásicas, por

ejemplo, gradient descent. El flujo se repite entonces de manera iterativa hasta que se alcanza el punto de convergencia. Es importante destacar que la fase de optimización requiere un ajuste apropiado para evitar caer en mínimos locales que evitan la convergencia. Existen estrategias para aliviar estos problemas, por ejemplo, la utilización de funciones de coste locales que solo incorporan información de parte del circuito, o el uso de técnicas más apropiadas al dominio como quantum natural gradient (Harrow & Napp, 2021) en lugar de gradient descent.

Los pasos 1 y 2 corresponden al bloque Quantum del diagrama, mientras que el paso 3 de optimización se incluye en el bloque Classical.

Una vez finalizado el proceso de optimización, se han obtenido los parámetros óptimos para el circuito variacional. Estos parámetros representan el ground state de la molécula, esto es, un eigenvector del Hamiltoniano, concretamente el eigenvector de mínima energía. Esto da lugar a una serie de aplicaciones de interés que se describen con mayor detalle en la siguiente subsección.

Además, se debe destacar que VQE tiene aplicaciones horizontales en otros dominios fuera del de la simulación química, por ejemplo, optimización. VQE es en definitiva un eigensolver, es decir, un método para el cálculo de los eigenvectors de un circuito cuántico (i.e. estados de mínima energía). Es decir, VQE es un algoritmo versátil que se identifica también como parte del conjunto de herramientas para la resolución de problemas de optimización junto con QAOA y FALQON (descritos en la siguiente sección).

Aplicaciones concretas

Las aplicaciones que se comentan a continuación emanan de los variational quantum algorithms en general, y VQE en particular:

- Una de las aplicaciones directas es la búsqueda del ground state de una molécula. El resultado de VQE describe la distribución de electrones en los orbitales que genera el estado de menor energía en la molécula.
- Derivado del punto anterior, otra aplicación de interés es la construcción de superficies de energía potencial (Potential Energy Surfaces o PES). Estas entidades describen la energía de la molécula en función de la posición de sus partículas. Esto permite estudiar, por ejemplo, energías de activación o energías de disociación de enlaces.
- Otra aplicación de interés es la simulación del proceso de plegamiento de proteínas, de acuerdo al proceso descrito en (Robert et al., 2021).
- Finalmente, se destaca la optimización de geometrías moleculares, es decir, el estudio de la distribución más estable de átomos en una molécula. Esta distribución viene dada por la configuración de menor energía de las posiciones de los átomos.

Optimización

En este dominio de optimización se pueden distinguir dos tipos de entidades principales:

- Las técnicas de optimización. Estas técnicas son métodos matemáticos para optimizar funciones de coste arbitrarias, es decir, permiten llevar a cabo el entrenamiento de circuitos

variacionales. Podría decirse que estas técnicas son intercambiables desde un punto de vista de interfaz de alto nivel (aunque en casos de uso reales no todas las técnicas resulten adecuadas). Es importante destacar que, tal y como se comentaba en la sección anterior de simulación, se basan en el procesamiento clásico. Algunas de ellas están específicamente diseñadas para su aplicación en el dominio de la computación cuántica, mientras que otras técnicas son bien conocidas y aplicables de manera general a otros dominios (por ejemplo, el machine learning).

- Los algoritmos de optimización. Estas son rutinas o métodos específicamente diseñados para resolver un subconjunto de problemas. Se apoyan sobre las técnicas de optimización previas. Suelen presentar arquitecturas y flujos de trabajo notablemente diferentes entre sí.

Los gradientes cuánticos son uno de los bloques básicos de construcción de estas técnicas y algoritmos de optimización. Los gradientes son derivadas parciales de la función cuántica que describe el valor de esperanza del circuito cuántico en cuestión. Las derivadas se calculan sobre los parámetros configurables de los circuitos, lo que enlaza con el concepto de circuito variacional que se comentaba anteriormente. Gracias a las propiedades matemáticas de estos sistemas, las derivadas (i.e. gradientes) se pueden descomponer en una combinación lineal de otras funciones cuánticas. Esta descomposición es, precisamente, lo que permite que los ordenadores cuánticos puedan calcular gradientes.

En esta sección se describen con más detalle dos algoritmos para la resolución de problemas de optimización combinatoria. Estos algoritmos se seleccionan inicialmente por su aplicabilidad a un rango amplio de problemas, incluyendo el de la optimización de rutas.

Es importante destacar que existen otras aportaciones a este campo de la optimización cuántica fuera de la órbita de QAOA. Por ejemplo, en (Wang & Xiang, 2019) los autores proponen un algoritmo comparable a Total Least Squares (TLS) adaptado al dominio de la computación cuántica. En este caso, TLS se reformula en términos de un Hamiltoniano cuyo estado fundamental (i.e. ground state) proporciona una solución al problema de ajuste por mínimos cuadrados; la solución es teóricamente obtenible de manera más eficiente en tiempo que con la alternativa clásica.

QAOA

Quantum Approximate Optimization Algorithm (QAOA) (Farhi et al., 2014) es un método popular y con mucha presencia dentro del dominio de la computación cuántica para la resolución de problemas de optimización combinatoria en dispositivos NISQ. Particularmente, QAOA resulta de especial utilidad para la resolución de problemas tradicionalmente difíciles en grafos (e.g. Travelling Salesman Problem).

Los circuitos cuánticos suelen definirse tradicionalmente como una secuencia de puertas cuánticas (e.g. Hadamard, CNOT). QAOA se asienta sobre la premisa de que un circuito cuántico también puede definirse en términos de un Hamiltoniano que describe su evolución a través del tiempo.

Un Hamiltoniano, como se comentaba en la sección de simulación química, representa la energía del circuito; en este caso el concepto de energía debe ser entendido desde un punto de vista más amplio para encajarlo en el dominio de la optimización.

La cuestión es que modelar Hamiltonianos complejos con puertas cuánticas en la forma exponencial de la ecuación anterior es significativamente complicado. Para solucionar este problema existen técnicas que permiten calcular el unitario U de manera aproximada en forma de una secuencia de bloques de puertas cuánticas, tal y como se muestra en la siguiente figura:

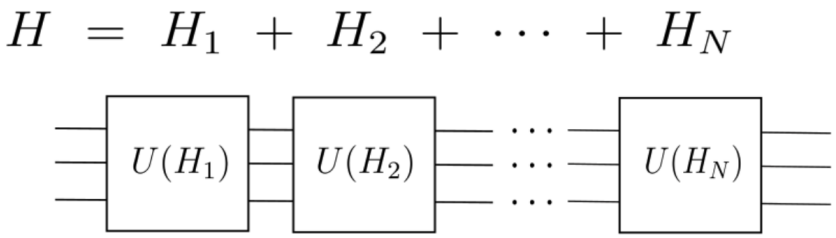


Figura 4 - Aproximación de modelado en forma de puertas cuánticas del unitario U en base a un Hamiltoniano H. Imagen extraída de (Ceroni, 2020).

El algoritmo QAOA hace uso de estos conceptos para proporcionar soluciones a retos de optimización combinatoria. De manera general, una instancia particular de QAOA viene definida por dos Hamiltonianos, un cost Hamiltonian (Hc) y un mixer Hamiltonian (Hm). Hc es el encargado de modelar el coste (la energía) del problema, de manera que el estado de menor energía del Hamiltoniano (i.e. ground state) sea una solución óptima a dicho problema.

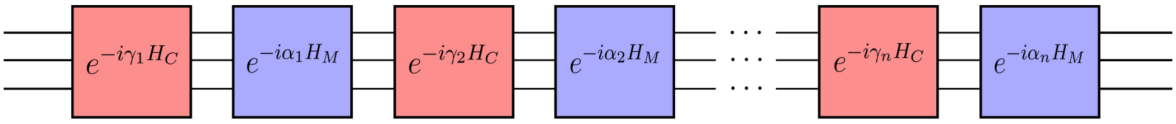


Figura 5 - Arquitectura de un circuito QAOA. Imagen extraída de (Ceroni, 2020).

La imagen anterior representa la arquitectura de un circuito QAOA. En ella se puede ver que se combina un número arbitrario de bloques compuestos por dos capas. La primera capa de un bloque se construye en base al cost Hamiltonian (Hc), mientras que la segunda capa se construye en base al mixer Hamiltonian (Hm). Los parámetros son en este caso el objetivo de optimización; es decir, la rutina de optimización tiene como resultado el conjunto de parámetros que producen un estado de salida del circuito cuántico que representa una solución aproximadamente óptima para el problema inicial.

Existe un ecosistema activo alrededor de QAOA, que ha dado lugar a una familia de algoritmos derivados, por ejemplo:

- Warm-start QAOA (Egger et al., 2021): La entrada al algoritmo QAOA es el resultado de una solución aproximada obtenida de manera computacionalmente eficiente con otro método.
- Quantum Alternating Operator Ansätze (Hadfield et al., 2021): Una generalización de QAOA que permite una mayor flexibilidad de parametrización y definición de restricciones.

FALQON

Feedback-Based Quantum Optimization (FALQON) (Magann et al., 2021) es una alternativa a QAOA para la resolución de problemas de optimización combinatoria. La principal diferencia con QAOA reside en que FALQON evita el uso de técnicas clásicas de optimización, lo que permite asegurar la convergencia del proceso de optimización. Es decir, FALQON no puede quedarse "atascado" en mínimos locales durante el proceso de optimización, cosa que sí puede ocurrir en QAOA. Por otra parte, FALQON tiene la desventaja de que requiere más recursos de computación porque la profundidad del circuito crece linealmente con el número de iteraciones de optimización.

Un detalle de interés relacionado con FALQON es que sus autores proponen la aproximación de combinar este algoritmo con QAOA. De esta manera, FALQON actúa como entrada a un circuito QAOA, es decir, los resultados del proceso de FALQON son la base para inicializar los parámetros bajo optimización en QAOA.

Como conclusión, se puede destacar que aunque FALQON no dispone del ecosistema y del volumen de referencias que se pueden encontrar en la órbita de QAOA, es igualmente una referencia interesante para evaluar en la implementación de soluciones de optimización basadas en computación cuántica. Especialmente en lo que se refiere a la combinación de FALQON con QAOA.

Aplicaciones concretas

QAOA, FALQON y otros algoritmos de optimización relacionados proporcionan, como se comentaba anteriormente, soluciones a problemas de optimización combinatoria. Concretamente, esto se traduce en la resolución de problemas NP-Hard en grafos. Por ejemplo:

- MAX-CUT: Búsqueda de una partición de nodos en dos conjuntos, de manera que se maximice el número de ejes conectados a nodos en dos conjuntos diferentes. Por ejemplo, este problema tiene aplicación concreta en el diseño de circuitos integrados.
- Minimum Vertex Cover: Búsqueda del conjunto de vértices con el menor tamaño posible de manera que los vértices están conectados a todos los nodos del grafo. Ejemplos de aplicación se encuentran en los campos de bioquímica o ciberseguridad.
- Maximum-Weighted Cycle: Búsqueda del ciclo del grafo con máximo coste (un ciclo comienza y termina en el mismo nodo, pero no repite ningún nodo en el camino intermedio). Este problema tiene aplicación, por ejemplo, en el dominio financiero y en la optimización de recursos e inversiones de manera general.

Clasificación

En esta sección se introducen modelos y técnicas orientados a tareas de clasificación predictiva. Estos presentan una alternativa en el dominio cuántico a modelos clásicos bien conocidos de Machine Learning, por ejemplo Logistic Regression y Support Vector Machines (SVM). De manera análoga, este dominio de conocimiento suele recibir el nombre de Quantum Machine Learning (QML).

Se puede argumentar que el ecosistema de computación cuántica en el dominio del QML es comparativamente más diverso que en los dominios de simulación y optimización. Una inspección inicial revela que VQE y QAOA se pueden identificar como los algoritmos más populares en el caso de simulación química y optimización respectivamente; es decir, un volumen significativo de los recursos gira en torno a esos algoritmos concretos. En el caso de QML, se encuentra un panorama

con mayor diversidad. Es por esto que en esta sección se incluyen referencias a varios modelos, que aunque persiguen objetivos similares, se diferencian en sus aproximaciones y estrategias concretas.

En la literatura se pueden encontrar trabajos previos orientados en reconstruir la arquitectura de redes neuronales clásicas en un contexto cuántico. Esto presenta una complejidad significativa, ya que las características de las redes neuronales, como por ejemplo las funciones de activación no lineales, no son exactamente trasladables a un circuito cuántico. Es por esto que a día de hoy una de las principales tendencias es la de explotar las posibilidades de circuitos cuánticos variacionales que pueden ser entrenados por técnicas clásicas. Un circuito variacional, como se comentaba en las secciones anteriores, es un circuito cuántico cuyos parámetros físicos (e.g. incidencia de un láser) pueden "entrenarse" a través de técnicas clásicas de optimización para que el circuito se ajuste a un problema concreto.

De manera estrechamente relacionada a los modelos descritos en esta sección, es necesario mencionar el conjunto de técnicas existentes para representación de las características de entrada (i.e. features). Estas técnicas reciben el nombre general de Quantum Embeddings, y permiten transformar un vector de datos clásico a una superposición de estados cuánticos que actúa como entrada. Se destacan dos estrategias de embeddings:

- Basis Embedding: Puede aplicarse en caso de que los vectores de entrada puedan describirse en términos de características binarias. Cada característica binaria se representa con un 1 o 0 en el qubit apropiado; por ejemplo, el vector de entrada [0,0,1] se representa como el estado |001>.
- Amplitude Embedding: Requiere un número igual o menor de qubits que la técnica de Basis Embedding para un vector de características de igual tamaño. Se basa en representar los elementos del vector clásico de características, previa normalización, en forma de diferentes amplitudes de la superposición de entrada.

La selección de modelos que se describe en mayor detalle a continuación representa una muestra significativa de las posibilidades que ofrece este dominio. De cualquier manera, se debe destacar que existen otras referencias de interés con aportaciones notables al campo del QML. Por ejemplo, el modelo Quantum Generative Adversarial Network (QGAN) de (Lloyd & Weedbrook, 2018), una aproximación cuántica al notable modelo clásico Generative Adversarial Network.

Quantum Convolutional Neural Networks

Las Quantum Convolutional Neural Networks (QCNN) (Cong et al., 2019) persiguen replicar la arquitectura de las populares Convolutional Neural Networks (CNN) utilizando piezas cuánticas.

Las CNN rinden particularmente bien en tareas relacionadas con la clasificación de imágenes, y son un componente fundamental de múltiples servicios de inteligencia artificial actualmente disponibles para el público en general. Se basan en la combinación secuencial de capas de convolución y pooling. De manera general, ambos tipos de capas se basan en la aplicación de un kernel para extracción de las características de alto nivel de la imagen (e.g. bordes) a la vez que se reduce la dimensión de los datos, de manera que sea computacionalmente viable trabajar con ellos. La etapa final de la arquitectura suele ser una capa plenamente conectada (fully connected) que permite el aprendizaje

de una función no lineal sobre las características calculadas por la secuencia de convolución y pooling.

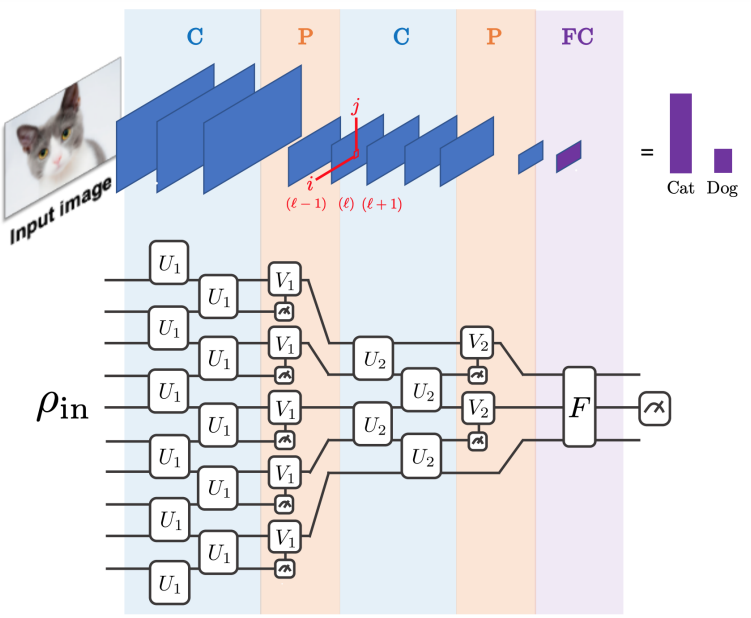


Figura 6 - Diagrama comparativo de arquitecturas CNN y QCNN. Imagen extraída de (Cong et al., 2019).

En el caso de las QCNN, las capas de convolución (C) se representan con circuitos variacionales, es decir, con secuencias de puertas cuánticas unitarias parametrizables que ejercen rotaciones sobre el vector de estado cuántico. Las capas de pooling (P) se representan con medidas de subconjuntos de los qubits, lo que reduce la dimensionalidad del vector, de manera análoga al caso clásico. Finalmente, la capa plenamente conectada (FC) se representa con otro circuito cuántico adicional que no reduce la dimensión del problema, es decir, que conserva el número de qubits.

Quanvolutional Neural Networks

El modelo Quanvolutional Neural Network (QNN) (Henderson et al., 2019) se inspira en los populares modelos clásicos Convolutional Neural Networks (CNN). Esto es de manera similar al anteriormente descrito modelo Quantum Convolutional Neural Network (QCNN). La diferencia principal reside en que QNN toma una aproximación híbrida, en la que se unen modelos cuánticos y clásicos, mientras que QCNN tiene una arquitectura puramente cuántica, en la que las capas de convolución y pooling se representan con puertas cuánticas.

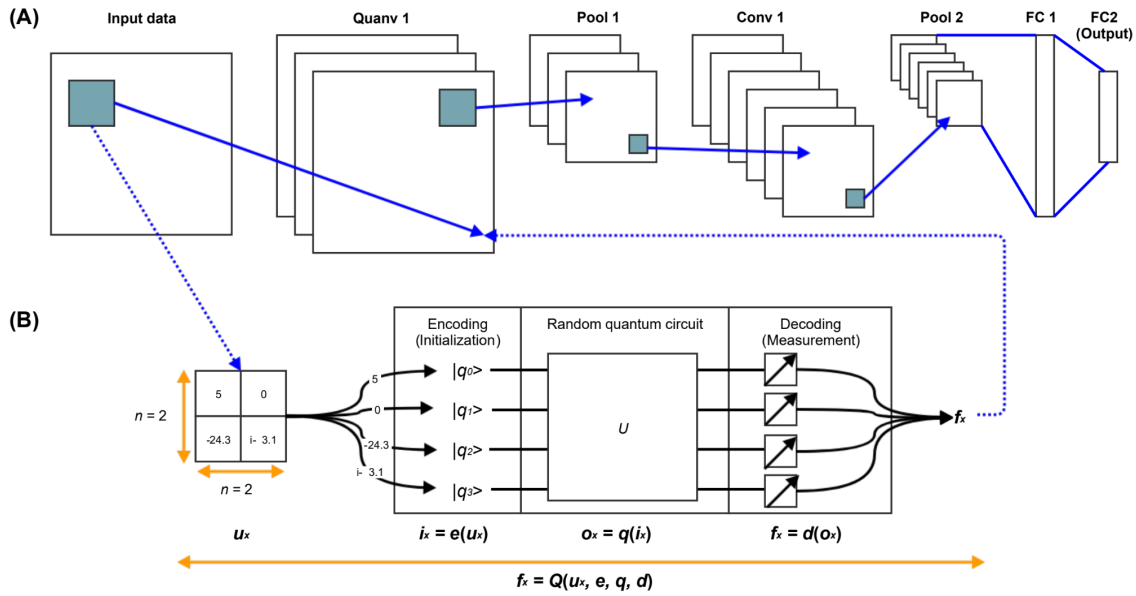


Figura 7 - Detalle de una capa quanvolutional (B) en el contexto de una arquitectura CNN clásica (A). Imagen extraída de (Henderson et al., 2019).

La idea principal detrás de las QNN es la de añadir una nueva herramienta a la arquitectura de las CNN en forma de la denominada capa quanvolutional. Una capa quanvolutional aplica un kernel "cuántico" a una región de la imagen, tal y como se muestra en la figura anterior. Es decir, los datos de imagen de entrada se codifican en un embedding cuántico al que luego se aplica un unitario para finalmente colapsar el statevector y obtener las medidas. Los parámetros del embedding del vector de datos clásico forman parte del proceso de entrenamiento de la red. Concretamente, la referencia sugiere la utilización de un circuito cuántico unitario aleatorio, aunque existen otras aproximaciones.

La aportación de mayor interés de esta aproximación es el hecho de que, gracias a la utilización de circuitos cuánticos, estas capas quanvolutional pueden aplicar kernels significativamente más complejos que sus análogos en computación clásica.

Quantum Transfer Learning

La técnica de Transfer Learning es una estrategia razonablemente popular que persigue aprovechar redes previamente entrenadas para problemas generales con buen rendimiento en el contexto de problemas particulares. De manera general, se eliminan las capas finales de la red generalista, para posteriormente añadir capas específicas que se someten a un nuevo proceso de entrenamiento. Los parámetros aprendidos de la red generalista permanecen estables.

El Transfer Learning permite, básicamente, obtener redes especializadas sin necesidad de invertir los significativos recursos de computación y diseño que requiere desarrollar una red desde cero con un nivel de rendimiento comparable.

La técnica de Quantum Transfer Learning (Mari et al., 2020) tiene muchos paralelismos con lo descrito anteriormente. Esta técnica propone la idea de utilizar una red neuronal basada en circuitos cuánticos, ya sea en lugar de la red generalista o de la red específica.

Se puede argumentar que la aproximación de utilizar una red clásica generalista y acoplar una red cuántica específica es la que resulta razonablemente más intuitiva. Es decir, en este caso se persigue aprovechar la probada calidad de una red bien conocida, proporcionándole un empuje de rendimiento gracias a la capacidades únicas de los sistemas cuánticos.

Con fines ilustrativos, se presenta a continuación un diagrama de una arquitectura de red de ejemplo para clasificación de imágenes basada en Quantum Transfer Learning. La red generalista en este caso es la bien conocida ResNet18, una red de 18 capas propuesta en (He et al., 2015) entrenada sobre el conjunto de datos ImageNet. A esta red se le elimina la última capa, que se sustituye por un circuito cuántico variacional (QPU) de 4 qubits encajado entre dos capas clásicas (L). La primera capa clásica reduce la dimensión de la penúltima capa de ResNet18 de 512 características a 4, mientras que la segunda capa clásica mapea a los dos elementos que representan la distribución de probabilidad de las clases de salida.

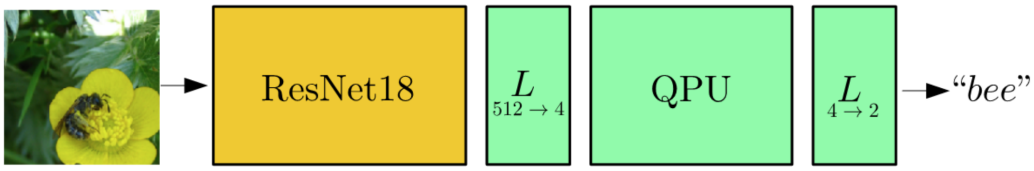


Figura 8 - Diagrama de un ejemplo concreto de implementación de Quantum Transfer Learning. Imagen extraída de (PennyLane dev team, 2021).

En (PennyLane dev team, 2021) se puede verificar la viabilidad de esta técnica de Quantum Transfer Learning en una implementación real y funcional del ejemplo previamente descrito.

Data Re-uploading Universal Classifier

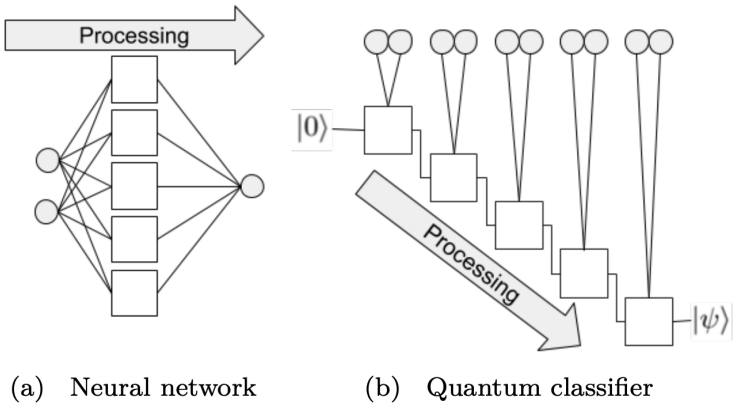
Los autores de (Pérez-Salinas et al., 2020) proponen la idea de que un solo qubit, en combinación con técnicas clásicas, proporciona suficientes capacidades para implementar modelos de clasificación viables. De hecho, los resultados que presentan en su estudio demuestran rendimientos comparables o mejores a modelos clásicos populares, concretamente SVM y una red neuronal con una sola capa oculta de 100 nodos.

La arquitectura de este modelo se basa en una secuencia de capas. Cada capa contiene dos puertas cuánticas capaces de aplicar una rotación arbitraria a un solo qubit. Esa rotación viene determinada por tres parámetros; los parámetros proceden a su vez de diferentes fuentes:

- Los parámetros de rotación de la primera puerta de una capa vienen determinados por el vector de datos de entrada. Puesto que son tres parámetros, se puede descomponer esta puerta en múltiples puertas para cubrir el caso de que la dimensión del vector de entrada sea mayor de tres. Se debe destacar que el vector de entrada se aplica en todas las capas, de esta idea precisamente viene el término de re-uploading.

- Los parámetros de rotación de la segunda puerta son los parámetros sujetos a entrenamiento. Es decir, en una arquitectura con una capa, hay tres parámetros a optimizar, con dos capas, hay seis parámetros a optimizar, y así sucesivamente.

Esta arquitectura se puede ver representada en la imagen a continuación:



(a) Neural network (b) Quantum classifier

Figura 9 - Arquitecturas de una red neuronal clásica (a) y una red Data Re-uploading Universal Quantum Classifier (b). Imagen extraída de (Pérez-Salinas et al., 2020).

Finalmente, se obtiene la esperanza de un observable cuántico aplicado al circuito en el que se codifica la etiqueta de la clase correcta. Esta esperanza describe la fidelidad a la clase correcta en cuestión. La función de coste se define entonces como un sumatorio de las fidelidades de todas las filas de entrada.

Variational Quantum Classifiers

La arquitectura de los modelos denominados como Variational Quantum Classifiers (VQC) emana de los trabajos (Farhi & Neven, 2018) y (Schuld et al., 2020).

Este tipo de arquitecturas se basan en una secuencia de capas reutilizables y parametrizables. Estas capas se definen usando puertas cuánticas que aplican rotaciones arbitrarias y puertas CNOT para permitir la creación de entrelazamientos. El quantum embedding de los vectores de entrada es un proceso razonablemente complejo que consiste, de manera general, en codificar la entrada en un conjunto de ángulos que actúan como parámetros de las puertas de rotación.

En este caso la función de coste se define en términos de computación clásica, de manera que proporcione la desviación entre las etiquetas resultantes del clasificador y las etiquetas correctas. El entrenamiento, por consiguiente, también se lleva a cabo con técnicas de computación clásica.

Este tipo de modelos presentan alguna incógnita más que alternativas comparables, pero apuntan a un alto nivel de adaptabilidad para distintos dominios de problema, por lo que resulta interesante considerarlos.

Aplicaciones concretas

Las posibilidades de aplicación del Quantum Machine Learning son significativamente amplias, de manera paralela a lo que ocurre con modelos clásicos tales como CNN, SVM o Logistic Regression. De hecho, el QML podría interpretarse como un añadido o herramienta complementaria al ML tradicional, proporcionando un empuje de rendimiento en un conjunto particular de casos.

Algunos ejemplos concretos incluyen la clasificación de correo basura, el reconocimiento de texto a partir de imagen, el análisis de sentimiento de mensajes de usuarios, la detección de caras y personas o la detección de actividades fraudulentas en operaciones financieras.

Referencias

- William John Munro, Victor M. Bastidas, Koji Azuma, & Kae Nemoto. (2021, May 5). Designing Quantum Computers. NTT Technical Review. https://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr202105fa4_s.html
- Farhi, E., Goldstone, J., Gutmann, S., & Sipser, M. (2000). Quantum Computation by Adiabatic Evolution. ArXiv:Quant-Ph/0001106. http://arxiv.org/abs/quant-ph/0001106
- Farhi, E., Goldstone, J., & Gutmann, S. (2014). A Quantum Approximate Optimization Algorithm. ArXiv:1411.4028 [Quant-Ph]. http://arxiv.org/abs/1411.4028
- Egger, D. J., Marecek, J., & Woerner, S. (2021). Warm-starting quantum optimization. Quantum, 5, 479. https://doi.org/10.22331/q-2021-06-17-479
- Hadfield, S., Hogg, T., & Rieffel, E. G. (2021). Analytical Framework for Quantum Alternating Operator Ans\"atze. ArXiv:2105.06996 [Quant-Ph]. http://arxiv.org/abs/2105.06996
- Magann, A. B., Rudinger, K. M., Grace, M. D., & Sarovar, M. (2021). Feedback-based quantum optimization. ArXiv:2103.08619 [Quant-Ph]. http://arxiv.org/abs/2103.08619
- Wang, H., & Xiang, H. (2019). Quantum algorithm for total least squares data fitting. Physics Letters A, 383(19), 2235–2240. https://doi.org/10.1016/j.physleta.2019.04.037
- Cao, Y., Romero, J., Olson, J. P., Degroote, M., Johnson, P. D., Kieferová, M., Kivlichan, I. D., Menke, T., Peropadre, B., Sawaya, N. P. D., Sim, S., Veis, L., & Aspuru-Guzik, A. (2019). Quantum Chemistry in the Age of Quantum Computing. Chemical Reviews, 119(19), 10856–10915. https://doi.org/10.1021/acs.chemrev.8b00803
- Harrow, A. W., & Napp, J. C. (2021). Low-depth gradient measurements can improve convergence in variational hybrid quantum-classical algorithms. Physical Review Letters, 126(14). https://doi.org/10.1103/physrevlett.126.140502
- Peruzzo, A., McClean, J., Shadbolt, P., Yung, M.-H., Zhou, X.-Q., Love, P. J., Aspuru-Guzik, A., & O'Brien, J. L. (2014). A variational eigenvalue solver on a photonic quantum processor. Nature Communications, 5(1), 4213. https://doi.org/10.1038/ncomms5213
- Ceroni, J. (2020, November 18). Intro to QAOA. PennyLane Demos. https://pennylane.ai/qml/demos/tutorial_qaoa_intro.html
- Cong, I., Choi, S., & Lukin, M. D. (2019). Quantum convolutional neural networks. Nature Physics, 15(12), 1273–1278. https://doi.org/10.1038/s41567-019-0648-8
- Mari, A., Bromley, T. R., Izaac, J., Schuld, M., & Killoran, N. (2020). Transfer learning in hybrid classical-quantum neural networks. Quantum, 4, 340. https://doi.org/10.22331/q-2020-10-09-340
- Henderson, M., Shakya, S., Pradhan, S., & Cook, T. (2019). Quanvolutional Neural Networks: Powering Image Recognition with Quantum Circuits. ArXiv:1904.04767 [Quant-Ph]. http://arxiv.org/abs/1904.04767

- Lloyd, S., & Weedbrook, C. (2018). Quantum generative adversarial learning. Physical Review Letters, 121(4), 040502. https://doi.org/10.1103/PhysRevLett.121.040502
- Farhi, E., & Neven, H. (2018). Classification with Quantum Neural Networks on Near Term Processors. ArXiv:1802.06002 [Quant-Ph]. http://arxiv.org/abs/1802.06002
- Schuld, M., Bocharov, A., Svore, K., & Wiebe, N. (2020). Circuit-centric quantum classifiers. Physical Review A, 101(3), 032308. https://doi.org/10.1103/PhysRevA.101.032308
- PennyLane dev team. (2021, January 28). Quantum transfer learning. PennyLane Demos. https://pennylane.ai/qml/demos/tutorial_quantum_transfer_learning.html
- He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. ArXiv:1512.03385 [Cs]. http://arxiv.org/abs/1512.03385
- Pérez-Salinas, A., Cervera-Lierta, A., Gil-Fuster, E., & Latorre, J. I. (2020). Data re-uploading for a universal quantum classifier. Quantum, 4, 226. https://doi.org/10.22331/q-2020-02-06-226
- Robert, A., Barkoutsos, P. Kl., Woerner, S., & Tavernelli, I. (2021). Resource-efficient quantum algorithm for protein folding. Npj Quantum Information, 7(1), 38. https://doi.org/10.1038/s41534-021-00368-4

T3.2: Identificación y selección de algoritmos de computación cuántica relacionados con clasificación, optimización y simulación

Esta tarea 3.2 se encuentra en proceso a finales de la anualidad 2021. Aun así, la tarea está cerca de su consecución, que está prevista para M13. La tarea tiene como objetivo la revisión de los algoritmos identificados en la tarea 3.1, de manera que se pueda seleccionar un conjunto definitivo que pasará a la fase de implementación. Para ello, se considerará principalmente la viabilidad de dicha implementación y los recursos disponibles en el ecosistema.

Las conclusiones *tentativas* en M12 se describen en los siguientes párrafos.

En el dominio de la **simulación química**, el algoritmo Variational Quantum Eigensolver (VQE) es el candidato más adecuado. Esta decisión se basa en una combinación del volumen de materiales asociados que están disponibles, el soporte en los frameworks software y la ausencia de algoritmos comparables. Es importante destacar que VQE también puede interpretarse como una familia de algoritmos, ya que se pueden identificar varias técnicas que son derivativas de VQE.

A continuación se muestra una prueba de concepto funcional de VQE en la que se persigue encontrar el estado fundamental de una molécula H2. Esta prueba de concepto tiene como objetivo último validar que la implementación de VQE es viable, y representa un segundo paso lógico después de la selección teórica del algoritmo.

```
== VQE: Prueba de concepto


+*In[1]:*+
[source, ipython3]
----
from pennylane import numpy as np
import pennylane as qml
----
```

Esta prueba de concepto se basa en una molécula muy simple con fines ilustrativos. Las coordenadas atómicas en este caso se definen manualmente, pero podrían obtenerse de un fichero de estructura `.xyz`.

El método `molecular_hamiltonian` proporcionado por PennyLane encapsula la lógica para construir el Hamiltoniano que permite representar la función de onda de la molécula.

+*In[2]:*+
[source, ipython3]
----
symbols = ["H", "H"]
coordinates = np.array([0.0, 0.0, -0.6614, 0.0, 0.0, 0.6614])
H, qubits = qml.qchem.molecular_hamiltonian(symbols, coordinates)

+*In[3]:*+
[source, ipython3]
----
print(f"Number of qubits = {qubits}")
print(f"Hamiltonian:\n{H}")

+*Out[3]:*+

Number of qubits = 4
Hamiltonian:
  (-0.2427450172749822) [Z2]
+ (-0.2427450172749822) [Z3]
+ (-0.04207254303152995) [I0]
+ (0.17771358191549907) [Z0]
+ (0.17771358191549919) [Z1]
+ (0.12293330460167415) [Z0 Z2]
+ (0.12293330460167415) [Z1 Z3]
+ (0.16768338881432715) [Z0 Z3]
+ (0.16768338881432715) [Z1 Z2]
+ (0.17059759240560826) [Z0 Z1]
+ (0.17627661476093917) [Z2 Z3]
+ (-0.04475008421265302) [Y0 Y1 X2 X3]
+ (-0.04475008421265302) [X0 X1 Y2 Y3]
+ (0.04475008421265302) [Y0 X1 X2 Y3]
+ (0.04475008421265302) [X0 Y1 Y2 X3]

+*In[4]:*+
[source, ipython3]
----
dev = qml.device("default.qubit", wires=qubits)

Cada átomo `H` tiene un electrón. Es decir, la molécula bajo prueba tiene *2 electrones en total*.

+*In[5]:*+
[source, ipython3]
----
electrons = 2

El método `hf_state` proporciona el estado cuántico inicial en función del número de electrones y qubits. De manera formal, el vector de ocupación que representa el estado Hartree-Fock.

+*In[6]:*+
[source, ipython3]
----
hf = qml.qchem.hf_state(electrons, qubits)

Seguidamente se define el circuito que se encarga de preparar el estado bajo prueba. Este tipo de circuitos se construyen a través de la combinación de puertas de rotación _Givens_. Es importante destacar que la arquitectura de puertas de rotación siempre debe asegurar la consistencia del estado de la molécula (i.e. _preserving the total-spin projection_). La lógica de construcción tiende a requerir una cierta dosis de ajuste manual, aunque existen aproximaciones automáticas que resultan menos óptimas (e.g. `qml.AllSinglesDoubles`).

+*In[7]:*+
[source, ipython3]
----
def circuit(param, wires):
    qml.BasisState(hf, wires=wires)
    qml.DoubleExcitation(param, wires=list(wires))

+*In[8]:*+
[source, ipython3]

@qml.qnode(dev)
def cost_fn(param):
    circuit(param, wires=range(qubits))
    return qml.expval(H)

+*In[9]:*+
[source, ipython3]
----
opt = qml.GradientDescentOptimizer(stepsize=0.4)

El parámetro _theta_ se debe inicializar a cero. Esto quiere decir que se parte del estado Hartree-Fock.

+*In[10]:*+
[source, ipython3]
---
theta = np.array(0.0, requires_grad=True)

A continuación se definen los parámetros que determinan el número máximo
de iteraciones de optimización y el valor del diferencial de energías
entre iteraciones que supone el límite en el que se asume que se ha
alcanzado la convergencia.

+*In[11]:*+
[source, ipython3]
---
max_iterations = 100
conv_tol = 1e-06

+*In[12]:*+
[source, ipython3]
---
energy = [cost_fn(theta)]
angle = [theta]

for n in range(max_iterations):
    theta, prev_energy = opt.step_and_cost(cost_fn, theta)
    curr_energy = cost_fn(theta)

    energy.append(curr_energy)
    angle.append(theta)

    conv = np.abs(curr_energy - prev_energy)

    if n % 2 == 0:
        print(f"Step = {n}, Energy = {energy[-1]:.8f} Ha")

    if conv <= conv_tol:
        break

+*Out[12]:*+

Step = 0, Energy = -1.12799983 Ha
Step = 2, Energy = -1.13466246 Ha
Step = 4, Energy = -1.13590595 Ha
Step = 6, Energy = -1.13613667 Ha
Step = 8, Energy = -1.13617944 Ha
Step = 10, Energy = -1.13618736 Ha
Step = 12, Energy = -1.13618883 Ha

Finalmente se ha conseguido llegar a la energía fundamental
(i.e. _ground state_) que coincide razonablemente con el valor esperado
que se puede calcular de manera clásica (véase la referencia

https://doi.org/10.1063/1.2747248).

+*In[13]:*+
[source, ipython3]
----
print(f"Ground state energy = {energy[-1]:.8f} Ha")
print(f"Optimal circuit parameter = {angle[-1]:.4f}")

+*Out[13]:*+

Ground state energy = -1.13618883 Ha
Optimal circuit parameter = 0.2089

De manera similar al caso anterior, en el dominio de la **optimización** se identifica Quantum Approximate Optimization Algorithm (QAOA) como el candidato final. Se deja la puerta abierta a combinar QAOA con FALQON, ya que existen referencias que apuntan a que esta combinación produce una mejora significativa de los resultados.

Dentro del proceso de selección del algoritmo QAOA, y de manera similar al caso anterior de VQE, se ha implementado una prueba de concepto minimalista para validación. La prueba de concepto persigue encontrar un *clique*, es decir, el subgrafo de mayor tamaño en el que todos los vértices están conectados. Este es un problema clásico de grafos que se clasifica bajo la categoría NP-complete.

== QAOA: Prueba de concepto

+*In[1]:*+
[source, ipython3]
----
from pennylane import qaoa
from pennylane import numpy as np
from matplotlib import pyplot as plt
import networkx as nx
import pennylane as qml

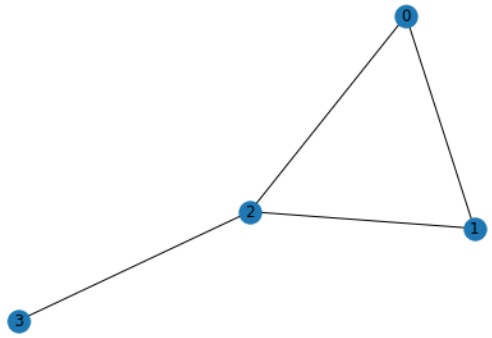
Primeramente se define un grafo sencillo con fines ilustrativos.

+*In[2]:*+
[source, ipython3]
----
edges = [(0, 1), (1, 2), (2, 0), (2, 3)]
graph = nx.Graph(edges)

+*In[3]:*+
[source, ipython3]

---
nx.draw(graph, with_labels=True)
plt.show()

+*Out[3]:*+



PennyLane proporciona un conjunto de métodos de gran utilidad que proporcionan definiciones automatizadas de los Hamiltonianos para un conjunto de problemas predefinidos. Por ejemplo, encontrar el *_clique_* de mayor tamaño en un grafo.

+*In[4]:*+
[source, ipython3]
---
cost_h, mixer_h = qaoa.max_clique(graph, constrained=True)

Un *_clique_* es un subgrafo para el que todos los vértices están conectados por un eje.

+*In[5]:*+
[source, ipython3]
---
print(f"Cost Hamiltonian:\n{cost_h}")
print(f"Mixer Hamiltonian:\n{mixer_h}")

+*Out[5]:*+

Cost Hamiltonian:
  (1) [Z0]
+ (1) [Z1]
+ (1) [Z2]
+ (1) [Z3]
Mixer Hamiltonian:
  (0.25) [X3]

```
+ (0.5) [X0]
+ (0.5) [X1]
+ (1.0) [X2]
+ (0.25) [X3 Z1]
+ (0.25) [X3 Z0]
+ (0.5) [X0 Z3]
+ (0.5) [X1 Z3]
+ (0.25) [X3 Z0 Z1]
----
```

```
+*In[6]:*+
[source, ipython3]
----
def qaoa_layer(gamma, alpha):
    qaoa.cost_layer(gamma, cost_h)
    qaoa.mixer_layer(alpha, mixer_h)
----
```

La profundidad del circuito QAOA se define manualmente.

```
+*In[7]:*+
[source, ipython3]
----
depth = 3
----
```

```
+*In[8]:*+
[source, ipython3]
----
wires = range(graph.number_of_nodes())

def circuit(params, **kwargs):
    assert len(params) == 2
    assert all(len(item) == depth for item in params)

    for w in wires:
        qml.Hadamard(wires=w)

    qml.layer(qaoa_layer, depth, params[0], params[1])
----
```

```
+*In[9]:*+
[source, ipython3]
----
dev = qml.device("default.qubit", wires=wires)

@qml.qnode(dev)
def cost_function(params):
    circuit(params)
    return qml.expval(cost_h)
----
```

El número de iteraciones de optimización es, al igual que la profundidad

del circuito `depth`, un parámetro manual.

+*In[10]:*+
[source, ipython3]
----
optimization_steps = 70

El valor de inicialización de los parámetros tiene relevancia en el sentido de que una selección desafortunada puede llevar a mínimos locales, es decir, evitar la convergencia de la optimización. La mejor aproximación en este caso parece ser llevar a cabo pruebas manuales con problemas similares de menor tamaño.

+*In[11]:*+
[source, ipython3]
----
params = np.array([
    [0.5] * depth,
    [0.5] * depth
], requires_grad=True)

+*In[12]:*+
[source, ipython3]
----
optimizer = qml.GradientDescentOptimizer()

for i in range(optimization_steps):
    params = optimizer.step(cost_function, params)

+*In[13]:*+
[source, ipython3]
----
print(f"Optimal parameters:\n{params}")

+*Out[13]:*+

Optimal parameters:
[[0.89178626 0.75037334 0.57462375]
 [0.20349048 0.81263752 0.73456243]]

+*In[14]:*+
[source, ipython3]

@qml.qnode(dev)
def probability_circuit(gamma, alpha):
    circuit([gamma, alpha])

```
    return qml.probs(wires=wires)

probs = probability_circuit(params[0], params[1])
----

El resultado óptimo en este caso deja fuera el vértice con índice *3*.
Es decir, la cuarta posición en la superposición del estado cuántico.
Esto es, efectivamente, el _clique_ del grafo de entrada.


+*In[15]:*+
[source, ipython3]
----
print(f"Optimal state: |{bin(probs.tolist().index(max(probs)))[2:]}>")
----


+*Out[15]:*+
----
Optimal state: |1110>
----
```

En el dominio de la **clasificación**, la decisión es más compleja. Existe un número mucho más elevado de algoritmos. Además, no se han encontrado referencias que proporcionen datos de rendimiento comparativos con un nivel de exhaustividad apropiado. Inicialmente, las técnicas de Quantum Transfer Learning y Data Re-Uploading Universal Classifier apuntan a ser las más interesantes por su viabilidad para ser implementadas en simuladores y ordenadores cuánticos NISQ razonablemente accesibles. Aun así, cerrar las puertas a otras técnicas, como por ejemplo las redes Quanvolutional, tendría demasiado riesgo. Es por esto que se recomienda no descartar ninguna técnica de antemano.

PT4: *Framework* de simulación cuántica

Este paquete de trabajo se organiza en torno a tres tareas, de las cuales una ha sido iniciada y finalizada en su totalidad en esta anualidad, y una segunda ha sido iniciada con previsión de finalización en el año 2022. La restante, se iniciará y finalizará a lo largo de la última anualidad.

- T4.1: Estudio de simuladores cuánticos
- T4.2: Integración de simuladores cuánticos en el hardware

A continuación, se describen las acciones desarrolladas en la tarea correspondiente por completo a esta anualidad y los principales resultados alcanzados. En el caso de la tarea iniciada, pero no finalizada, se dará una breve descripción de los desarrollos realizados hasta el momento y que se presentarán de forma finalizada en la próxima anualidad.

T4.1: Estudio de simuladores cuánticos

El contenido relativo a esta tarea presenta una introducción a las características de un conjunto de frameworks software que ofrecen funcionalidades en el dominio de la computación cuántica. Estos frameworks han sido escogidos en función de su popularidad en las referencias de la literatura y el peso de las organizaciones que los promueven.

Se describen las funcionalidades más interesantes de cada uno de los frameworks. Seguidamente, se muestra una comparativa de pruebas de concepto básicas para demostración de las interfaces programáticas (API). Finalmente, se incluye una sección de conclusiones en la que se toma la decisión final de utilización de librería.

Aunque no se ha incluido en el proceso de análisis profundo, es importante destacar también la existencia del Quantum Development Kit de Microsoft, con mención especial a Q#, un nuevo lenguaje de programación para el diseño de circuitos cuánticos. Esta tecnología no se incluye en el análisis porque se estimó que las diferencias principales residían en su aproximación, es decir, las funcionalidades ofrecidas eran comparables a los otros tres candidatos, por lo que podía resultar redundante.

Frameworks de simulación cuántica

Qiskit

Qiskit (ANIS et al., 2021) es un framework desarrollado por IBM que aspira a proporcionar una solución completa para flujos de trabajo en el dominio de la computación cuántica. Está compuesto por una serie de módulos que implementan diferentes niveles de abstracción, desde los componentes básicos de circuitos cuánticos hasta algoritmos completos reutilizables.

La organización de Qiskit se asienta en un conjunto de módulos centrales que son los bloques básicos de construcción y simulación de circuitos. Encima de este núcleo se encuentran un conjunto de módulos de dominio específico para aplicaciones más avanzadas.

Qiskit permite ejecutar circuitos cuánticos en ordenadores cuánticos reales a través de la interfaz del IBM Quantum Provider. Este módulo encapsula los detalles de bajo nivel de la comunicación, y habilita un flujo de trabajo ágil en el que se puede iterar rápidamente entre simulación y ejecución real utilizando las mismas herramientas proporcionadas por Qiskit. Obviamente, existen restricciones de uso gratuito de los ordenadores cuánticos.

Qiskit Terra contiene los cimientos del framework Qiskit. Es decir, todos los programas desarrollados sobre Qiskit utilizan Terra de manera directa o indirecta. Concretamente, Terra encapsula de manera principal las siguientes funcionalidades y entidades:

- Construcción de circuitos cuánticos a través de la combinación de puertas cuánticas. Qiskit ofrece un catálogo amplio de puertas, incluyendo todos los tipos de puerta de relevancia en el estado de la técnica.
- La abstracción de Providers y Backends que exponen el acceso a dispositivos cuánticos externos. Un Provider actúa como punto único de entrada a un conjunto de Backends, gestionando normalmente la autenticación. Los Backends representan a su vez dispositivos específicos de simulación cuántica o computadores cuánticos reales.
- Transpiladores de circuitos cuánticos para su adaptación a las particularidades de un Backend específico (e.g. un ordenador cuántico). Los transpiladores se encargan de llevar a cabo las transformaciones necesarias para asegurar que un circuito se puede ejecutar en un Backend. Aunque Qiskit permite la definición de circuitos en términos de puertas de alto nivel, es común que un dispositivo cuántico presente restricciones en el tipo de puertas y

configuraciones que se pueden ejecutar, así que algunos fragmentos de los circuitos deben descomponerse en puertas cuánticas más simples.

- Herramientas para la visualización del estado de circuitos cuánticos. Por ejemplo, representaciones del estado de un qubit en la Bloch Sphere, o del estado de un circuito en una Q-sphere (una representación particular de Qiskit).
- Una abstracción para encapsulado de experimentos y configuración denominada Qobj. El Qobj resulta especialmente útil para asegurar la reproducibilidad de los experimentos y facilitar su transporte.

En la figura a continuación se muestra un ejemplo de visualización sobre IBM Quantum Composer, una aplicación Web para simplificar el diseño de circuitos cuánticos que está desarrollada por IBM y basada en Qiskit.

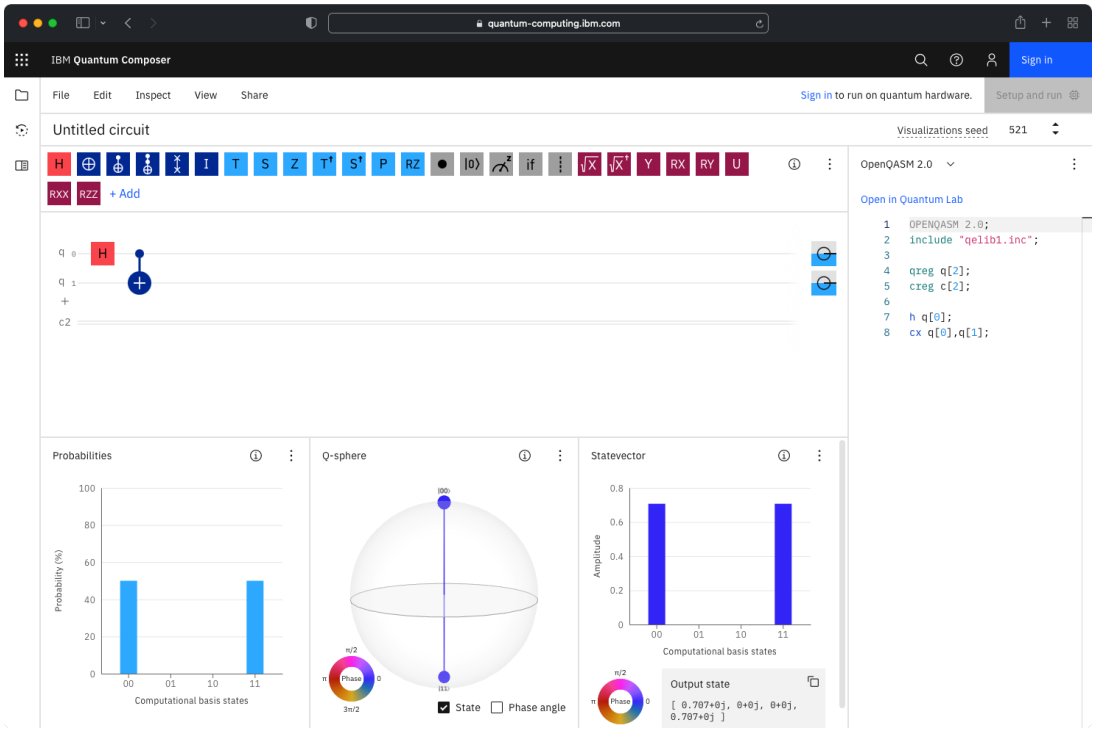


Figura 10 - Construcción de un circuito cuántico para la definición de un estado entrelazado (Bell State) en IBM Quantum Composer.

Qiskit Aer expone simuladores cuánticos de alto rendimiento, así como las entidades necesarias para añadir modelos de ruido realistas a las simulaciones. Estos modelos de ruido pueden capturarse en los ordenadores cuánticos reales de IBM, lo que permite acercarse más a condiciones representativas de la realidad.

Aer es una herramienta indispensable para la simulación y validación de circuitos cuánticos en el contexto de Qiskit. De manera general, un simulador cuántico es capaz de desarrollar los cálculos de álgebra lineal que determinan el comportamiento de un circuito cuántico. Gracias a esto se puede calcular, por ejemplo, el estado del vector cuántico (statevector) o la matriz unitaria (unitary) que representa al circuito.

Finalmente, es importante destacar que los antiguos módulos Qiskit Aqua y Qiskit Ignis han pasado a estado deprecated. La mayoría de su funcionalidad se ha reorganizado en otros componentes de Qiskit. Ignis contenía modelos para caracterización de ruido y corrección de errores, mientras que Aqua contenía un conjunto pre-empaquetado de algoritmos y plantillas de circuitos.

Adicionalmente, Qiskit ofrece un conjunto de módulos de dominio específico construidos sobre los módulos centrales. Cada uno de ellos está enfocado en una área concreta de aplicación. La tabla a continuación presenta un resumen de las características y algoritmos más destacables.

Tabla. Módulos de dominio específico de Qiskit.

Módulo	Descripción	Algoritmos destacados
Machine Learning	Utilidades para el diseño de modelos de machine learning basados en computación cuántica, es decir, modelos dentro del dominio de Quantum Machine Learning (QML). En este caso los parámetros de los circuitos cuánticos se someten a procesos de optimización que permiten "entrenar" el circuito para ajustarse a un problema concreto, de manera similar al machine learning clásico. La mayoría de los modelos tienden a estar inspirados por equivalentes en el dominio clásico (e.g. SVM).	• Quantum Support Vector Classifier (QSVC). • Variational Quantum Classifier (VQC). • Quantum Generative Adversarial Network (QGAN).
Nature	Algoritmos cuánticos aplicables, principalmente, a los dominios de la química y la física. Además, contiene la funcionalidad necesaria para definir de manera programática la estructura de partículas en términos cuánticos. Ejemplos de problemas concretos solucionables con este módulo son el cálculo de Potential Energy Surfaces (PES), hallar el ground state de energía de una molécula, y la simulación de procesos de plegado de proteínas.	• Adaptive Variational Quantum Eigensolver (AdaptVQE). • Born-Oppenheimer Potential Energy Surface (BOPES).
Finance	Algoritmos para solución de problemas en el dominio de las finanzas. Por ejemplo, optimización de portfolios (i.e. distribución óptima de un conjunto de activos entre productos financieros) o análisis de riesgos.	Este módulo contiene implementaciones de casos de uso (e.g. optimización de portfolio) pero no expone ningún algoritmo de dominio específico.

Optimization	Definición de problemas de optimización de alto nivel para la optimización de funciones de coste arbitrarias. Esto incluye ejemplos de problemas concretos tales como MAX-CUT, Minimum Vertex Cover, Travelling Salesman (TSP) o problemas tipo Quadratic Unconstrained Binary Optimization (QUBO) en general.	• Warm-start Quantum Approximate Optimization Algorithm (QAOA) (QAOA está contenido en Qiskit Terra). • Grover Adaptive Search (GAS).

Cirq

Cirq (Cirq Developers, 2021) es la librería para diseño y simulación de circuitos cuánticos dentro del ecosistema de Google Quantum AI. A continuación se puede ver un diagrama de alto nivel de este ecosistema. En los siguientes párrafos de esta sección se verán en mayor detalle los elementos más relevantes de Google Quantum AI.

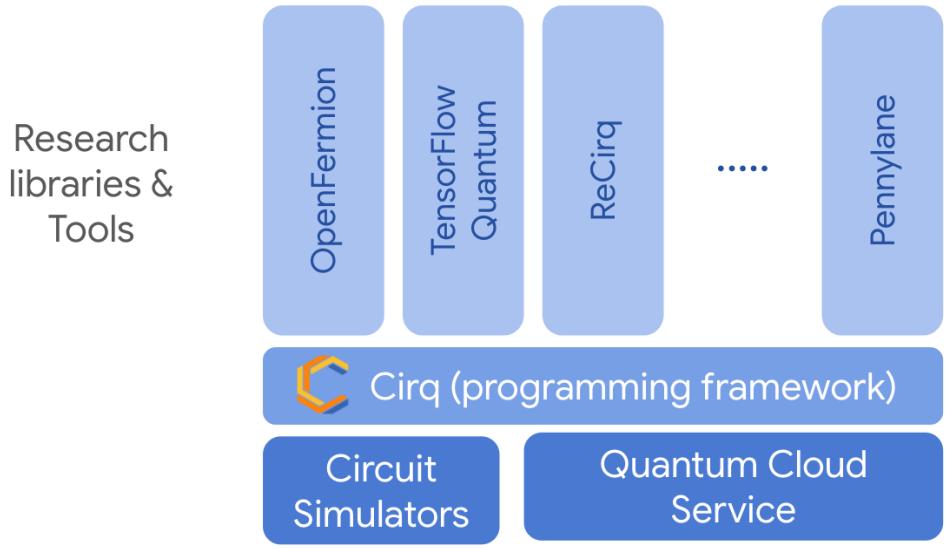


Figura 11 - Diagrama de ecosistema de Cirq. Imagen extraída de (Cirq Developers, 2021).

Cirq expone una API razonablemente similar a Qiskit que está basada en las entidades de un circuito cuántico: Circuits, que representan una configuración específica de qubits y puertas cuánticas; y Gates, que pueden ser aplicadas a Qubits para generar Operations. A un conjunto de operaciones que actúa en el mismo momento temporal se le conoce como un Moment. Como podía esperarse, Cirq viene integrado con un catálogo de puertas cuánticas comunes de uno y dos qubits, por ejemplo, puertas de rotación en los ejes X, Y, Z, y la puerta CNOT.

Cirq también tiene librerías o módulos externos compatibles que contienen algoritmos de dominio específico, como se puede ver en la siguiente tabla:

Tabla. Librerías de dominio específico en el ecosistema Google Quantum AI.

Módulo	Descripción
Tensorflow Quantum (Broughton et al., 2021)	Una extensión del popular framework Tensorflow enfocado en el dominio del Quantum Machine Learning (QML). De manera más específica, Tensorflow Quantum incorpora la computación cuántica en forma de Quantum Processing Units (QPU) al conjunto ya existente de infraestructura CPU, GPU y TPU que existe en Tensorflow clásico. La principal ventaja de Tensorflow Quantum es precisamente esta, poner los excelentes recursos actualmente existentes para Keras y Tensorflow a disposición del dominio del QML. Según sus autores, Tensorflow Quantum tiene dos objetivos principales. Primeramente, usar modelos clásicos de machine learning para mejorar el rendimiento de algoritmos cuánticos como QAOA. En segundo lugar, aprovechar las características de los ordenadores cuánticos para modelar información cuántica en procesos de machine learning, lo que resulta muy complejo usando solamente recursos clásicos.
OpenFermion (McClean et al., 2020)	Una librería para la simulación de sistemas fermiónicos a través de circuitos cuánticos. Es decir, que resulta aplicable al dominio de la química cuántica. Permite definir Hamltonianos en términos de qubits o fermiones, e incluye las transformaciones y puertas cuánticas necesarias para operar en estos dos dominios. Concretamente, para el caso de fermiones existe un simulador de alto rendimiento llamado Fermionic Quantum Emulator (FQE) (Nicholas C. Rubin, Klaas Gunst, 2021).

El simulador por defecto incluido en Cirq (cirq.Simulator) está limitado a 20 qubits. Se puede llegar sin embargo hasta 40 qubits con Cirq utilizando qsim, un simulador de función de onda de alto rendimiento dentro del ecosistema (Quantum AI Team & Collaborators, 2020). Otra alternativa destacable es qFlex, que puede resultar más adecuado para circuitos "poco profundos", es decir, que presentan un alto número de qubits pero tienen pocas puertas cuánticas.

De manera similar a Qiskit, Cirq permite ejecutar los circuitos en ordenadores cuánticos reales sobre el servicio de Quantum Computing Service utilizando la interfaz de Quantum Engine API (encapsulada en el módulo cirq_google.engine.Engine). A día de hoy existe una limitación significativa en comparación con Qiskit, ya que la Quantum Engine API no está abierta para acceso público. Por otra parte, se espera que el acceso se abra en un futuro cercano. Adicionalmente, aunque se puede argumentar que el Quantum Computing Service de Google es el servicio más relevante para ejecución real con Cirq, se debe destacar que Cirq tiene compatibilidad con otros servicios como Alpine Quantum Technologies y Microsoft Azure Quantum.

PennyLane

Al igual que en los casos anteriores, PennyLane (Bergholm et al., 2020) es una librería de código abierto que expone funcionalidad en diferentes niveles de abstracción para la construcción y simulación de circuitos cuánticos.

Una de las diferencias más significativas es que PennyLane puede actuar como capa de alto nivel que permite la utilización de los otros frameworks de manera transparente. De hecho, se puede ver que la figura del ecosistema de Cirq de la sección anterior presenta PennyLane como un componente de capa alta dentro de Google Quantum AI. El equipo de desarrollo de PennyLane ha invertido esfuerzos significativos en conseguir que PennyLane sea una capa de alto nivel agnóstica a la plataforma hardware y las librerías software.

Otra diferencia importante es la aproximación al diseño de la API. En Qiskit y Cirq los circuitos se inicializan explícitamente y se modifican de manera imperativa. En PennyLane los circuitos cuánticos (i.e. circuitos variacionales) toman la forma de funciones que están enlazadas a una interfaz y un dispositivo (más información a continuación). Dentro de la función se definen las puertas cuánticas que se aplican al circuito, así como la lógica adicional que sea necesaria para su construcción. PennyLane proporciona un decorador específico para este tipo de funciones (qml.qnode). Esta abstracción recibe el nombre de QNode, como se puede ver en el siguiente diagrama:

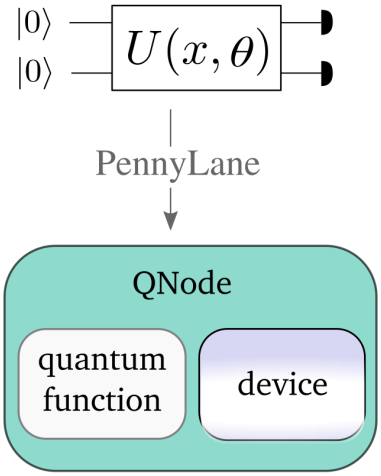


Figura 12 - Diagrama QNode. Imagen extraída de (PennyLane dev team, 2021).

Para la construcción de modelos híbridos clásico-cuánticos, PennyLane incorpora actualmente cuatro interfaces de integración con librerías de computación clásica. Las integraciones habilitan que las librerías se comuniquen con los circuitos cuánticos, es decir:

- Traducen las entidades nativas clásicas (e.g. NumPy ndarrays) a parámetros de los circuitos cuánticos en PennyLane.
- Habilitan que el circuito cuántico pueda usar técnicas de optimización clásicas a través del cálculo de gradientes.

Las librerías clásicas para las que existen interfaces son las siguientes:

Tabla. Interfaces incluidas en PennyLane.

Interfaz	Descripción
NumPy	Esta interfaz está basada en autograd, un paquete para el cálculo de derivadas en el ecosistema de NumPy. Es la interfaz por defecto de PennyLane, probablemente, la que tiene mejor documentación y soporte.
JAX	Autograd no se encuentra actualmente en desarrollo activo. JAX representa la nueva versión de Autograd. La diferencia principal entre ambos es que JAX utiliza Accelerated Linear Algebra (XLA), un compilador específico para deep learning, con el objetivo de acelerar los cálculos basados en álgebra lineal sobre GPU y Tensor Processing Units (TPU).
Tensorflow	Una popular plataforma respaldada por Google que ofrece extensa funcionalidad para diseño, entrenamiento y despliegue de modelos de machine learning.
PyTorch	Una de las alternativas a Tensorflow más relevantes. PyTorch está respaldado por Facebook. Las funcionalidades de ambas plataformas son comparables (e.g. entrenamiento distribuido, despliegue escalable de modelos).

Sobre estas interfaces, PennyLane define a su vez un conjunto compatible de optimizadores (optimizers) específicos para cada interfaz. Los optimizadores se encargan de actualizar y buscar parámetros óptimos de modelos híbridos clásico-cuánticos. El proceso de optimización se basa, a grandes rasgos, en el cálculo de gradientes, es decir, en la capacidad de calcular las derivadas de circuitos variacionales cuánticos.

De manera paralela, los dispositivos (devices) son las plataformas de computación sobre las que se ejecutan los circuitos cuánticos. Un dispositivo puede representar un simulador cuántico en un ordenador clásico, o un ordenador cuántico real. De hecho, el mismo flujo PennyLane puede combinar distintos dispositivos. Además, cualquier interfaz puede combinarse arbitrariamente con cualquier dispositivo. PennyLane incorpora de serie soporte para seis dispositivos, y además dispone de acceso a un conjunto amplio de plugins. Algunos ejemplos destacables de dispositivos son:

Tabla. Ejemplos de devices disponibles en PennyLane.

Device	Distribución	Descripción
default.qubit	Built-in	Simulador simple escrito en Python para circuitos cuánticos de medio y bajo volumen.
default.qubit.tf	Built-in	Simulador implementado sobre Tensorflow apropiado para circuitos cuánticos de mayor volumen.

Qiskit	Plugin	Habilita el acceso al simulador de alto rendimiento Aer y a IBM Quantum Provider para computación cuántica real.
Cirq	Plugin	Habilita el acceso al simulador por defecto de Cirq y a la extensión de simulación de alto rendimiento qsim.

PennyLane pone el foco en el cálculo de gradientes, y consecuentemente, en la optimización de circuitos variacionales. Es decir, PennyLane es principalmente una librería para la construcción de modelos de QML, para lo cual proporciona una documentación extensa y de muy alta calidad. Además, también incorpora un módulo qchem para el diseño de programas en el dominio de la química cuántica. PennyLane puede integrarse con PySCF, Psi4 y OpenFermion (este último parte de Google Quantum AI). Esto le permite, por ejemplo, soportar la construcción de Hamiltonianos en términos de qubits, la simulación de funciones de onda, el cálculo de ground states y el modelado de Potential Energy Surfaces (PES).

Comparativa de API

Esta sección presenta una breve comparativa de las interfaces (i.e. APIs) de los frameworks descritos anteriormente. Esta comparativa resulta interesante porque una interfaz bien diseñada, que exponga abstracciones de alto nivel, suele resultar en una mayor productividad. Por el contrario, las interfaces obtusas o complejas pueden presentar barreras notables en el proceso de desarrollo.

En la comparativa se ha programado una prueba de concepto muy simple que consiste en la definición de un circuito cuántico que configura el conocido como Bell State (un estado entrelazado). Una vez hecho esto, se imprime el diagrama del circuito y se simula el statevector de salida.

Los programas que se muestran a continuación son la representación en formato Asciidoc de un conjunto de notebooks Jupyter.

Programa desarrollado con la API de **Qiskit**:

```
+*In[1]:*+
[source, ipython3]
----
from qiskit import QuantumCircuit, Aer
----


+*In[2]:*+
[source, ipython3]
----
qiskit_circ = QuantumCircuit(2)
qiskit_circ.h(0)
qiskit_circ.cx(0, 1)
qiskit_circ.draw()
----


+*Out[2]:*+
```

```
----
....
   ┌─────┐
q_0:┤ H ├──■──
   └─────┘┌─┴─┐
q_1:──────┤ X ├
          └───┘
....
----


+*In[3]:*+
[source, ipython3]
----
qiskit_backend = Aer.get_backend("statevector_simulator")
qiskit_job = qiskit_backend.run(qiskit_circ)
qiskit_result = qiskit_job.result()
qiskit_statevector = qiskit_result.get_statevector(qiskit_circ, decimals=3)
----


+*In[4]:*+
[source, ipython3]
----
qiskit_statevector
----


+*Out[4]:*+
----
Statevector([0.707+0.j, 0.  +0.j, 0.  +0.j, 0.707+0.j],
            dims=(2, 2))
----
```

Programa desarrollado con la API de **Cirq**:

```
+*In[5]:*+
[source, ipython3]
----
import cirq
import cirq_google
----


+*In[6]:*+
[source, ipython3]
----
cirq_circuit = cirq.Circuit()
cirq_q0, cirq_q1 = cirq.LineQubit.range(2)
cirq_circuit.append(cirq.H(cirq_q0))
cirq_circuit.append(cirq.CNOT(cirq_q0, cirq_q1))
----


+*In[7]:*+
```

```
[source, ipython3]
----
cirq_circuit
----


+*Out[7]:*+
----
....
0: ———H———@———
          |
1: ———————X———
....
----


+*In[8]:*+
[source, ipython3]
----
cirq_sim = cirq.Simulator()
results = cirq_sim.simulate(cirq_circuit)
----


+*In[9]:*+
[source, ipython3]
----
results
----


+*Out[9]:*+
----measurements: (no measurements)
output vector: 0.707|00⟩ + 0.707|11⟩----
```

Programa desarrollado con la API de **PennyLane**:

```
+*In[10]:*+
[source, ipython3]
----
import pennylane as qml
----


+*In[11]:*+
[source, ipython3]
----
qml_dev = qml.device("default.qubit", wires=2)
----


+*In[12]:*+
[source, ipython3]
----
@qml.qnode(qml_dev)
```

```
def pennylane_circuit():
   qml.Hadamard(wires=0)
   qml.CNOT(wires=[0, 1])
   return qml.state()
----


+*In[13]:*+
[source, ipython3]
----
print(qml.draw(pennylane_circuit)())
----


+*Out[13]:*+
----
 0: ──H──╭C──┤  State
 1: ─────╰X──┤  State

----


+*In[14]:*+
[source, ipython3]
----
pennylane_circuit()
----


+*Out[14]:*+
----tensor([0.70710678+0.j, 0.       +0.j, 0.       +0.j, 0.70710678+0.j], requires_grad=True)----
```

Conclusiones

En esta sección se presentan las conclusiones de alto nivel extraídas de este proceso de revisión de frameworks de simulación cuántica.

- Las interfaces (APIs) de las opciones consideradas son razonablemente similares. Todas ellas siguen patrones de diseño modernos y resultan ágiles en el desarrollo. Qiskit tiene algo más de funcionalidad en el plano de la visualización y las herramientas de soporte. PennyLane se destaca utilizando una aproximación basada en decoradores para la definición de circuitos cuánticos, lo cual podría facilitar la encapsulación y la limpieza del código.
- cuQuantum, un SDK de NVIDIA para aceleración de flujos de computación cuántica utilizando GPUs, es una de las tendencias con mayor potencial para mejorar significativamente el tamaño de los problemas abordables. Tanto Cirq como Qiskit tienen integración con cuQuantum, lo que resulta un factor importante a tener en cuenta.
- Cirq se encuentra actualmente en estado alpha. Los otros dos candidatos, especialmente Qiskit, parecen encontrarse en un estado más estable. Sin embargo, una inspección de Cirq revela que la categorización alpha podría ser excesivamente conservadora, ya que existe un alto volumen de materiales de referencia de calidad para Cirq.
- Qiskit tiende a destacar en términos de funcionalidad incluida en el paquete, por ejemplo, en la implementación de plantillas para algoritmos bien conocidos. Aunque esta funcionalidad

adicional puede no ser significativa en una mayoría de los casos, es interesante tenerla en cuenta.

- OpenFermion, la herramienta para simulación de química cuántica de Cirq, proporciona abstracciones de menor nivel que PennyLane. De hecho, PennyLane construye su módulo de química cuántica sobre las funcionalidades de OpenFermion. Es decir, en PennyLane se demanda menor conocimiento experto del investigador, lo que puede facilitar el desarrollo de soluciones en este dominio.
- PennyLane, aunque carece del respaldo directo de un gigante tecnológico como ocurre en el caso de Cirq (Google) o Qiskit (IBM), permite el acceso a una amplia variedad de dispositivos cuánticos reales y simuladores. Esto permite maximizar la flexibilidad para seleccionar la plataforma más adecuada en función del problema concreto.

La conclusión final es que ninguna de las librerías consideradas destaca claramente sobre las demás. Además, hay un nivel significativo de superposición en sus funcionalidades. Qiskit puede resultar un producto más maduro y accesible, mientras que PennyLane es muy interesante y flexible gracias a su diseño agnóstico, finalmente, Cirq ofrece recursos de calidad orientados a la investigación, así como integración de primer nivel de Tensorflow Quantum. Descartar de antemano alguna de estas opciones es un riesgo innecesario.

La estrategia propuesta a partir de este punto se basará en usar PennyLane como opción por defecto, debido a su flexibilidad, sin perder de vista las posibilidades ofrecidas por Qiskit o Cirq. Estos últimos podrían cubrir deficiencias descubiertas en el proceso de los trabajos futuros con PennyLane.

Referencias

- Cirq Developers. (2021). Cirq (v0.12.0) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.5182845
- ANIS, M. S., Abraham, H., AduOffei, Agarwal, R., Agliardi, G., Aharoni, M., Akhalwaya, I. Y., Aleksandrowicz, G., Alexander, T., Amy, M., Anagolum, S., Arbel, E., Asfaw, A., Athalye, A., Avkhadiev, A., Azaustre, C., BHOLE, P., Banerjee, A., Banerjee, S., … Čepulkovskis, M. (2021). Qiskit: An open-source framework for quantum computing. https://doi.org/10.5281/zenodo.2573505
- Bergholm, V., Izaac, J., Schuld, M., Gogolin, C., Alam, M. S., Ahmed, S., Arrazola, J. M., Blank, C., Delgado, A., Jahangiri, S., McKiernan, K., Meyer, J. J., Niu, Z., Száva, A., & Killoran, N. (2020). PennyLane: Automatic differentiation of hybrid quantum-classical computations. ArXiv:1811.04968 [Physics, Physics:Quant-Ph]. http://arxiv.org/abs/1811.04968
- Quantum AI Team & Collaborators. (2020). Qsim. Zenodo. https://doi.org/10.5281/zenodo.4023103
- Cirq Developers. (2021, November 11). Research libraries and tools. Cirq Guide. https://quantumai.google/cirq/ecosystem
- Broughton, M., Verdon, G., McCourt, T., Martinez, A. J., Yoo, J. H., Isakov, S. V., Massey, P., Halavati, R., Niu, M. Y., Zlokapa, A., Peters, E., Lockwood, O., Skolik, A., Jerbi, S., Dunjko, V., Leib, M., Streif, M., Von Dollen, D., Chen, H., … Mohseni, M. (2021). TensorFlow Quantum: A Software Framework for Quantum Machine Learning. ArXiv:2003.02989 [Cond-Mat, Physics:Quant-Ph]. http://arxiv.org/abs/2003.02989
- McClean, J. R., Rubin, N. C., Sung, K. J., Kivlichan, I. D., Bonet-Monroig, X., Cao, Y., Dai, C., Fried, E. S., Gidney, C., Gimby, B., Gokhale, P., Häner, T., Hardikar, T., Havlíček, V., Higgott, O., Huang,

C., Izaac, J., Jiang, Z., Liu, X., … Babbush, R. (2020). OpenFermion: The electronic structure package for quantum computers. Quantum Science and Technology, 5(3), 034014. https://doi.org/10.1088/2058-9565/ab8ebc
- Nicholas C. Rubin, Klaas Gunst, A. W., Leon Freitag, Kyle Throssell, Garnet Chan, Ryan Babbush, Toru Shiozaki. (2021). The fermionic quantum emulator, version 0.2.0. https://github.com/quantumlib/OpenFermion-FQE
- PennyLane dev team. (2021, November). Quantum circuits. https://pennylane.readthedocs.io/en/stable/introduction/circuits.html

T4.2: Integración de simuladores cuánticos en el hardware

La tarea 4.2 comprende los trabajos de integración de un subconjunto de los frameworks software identificados en 4.1 en el contexto de una infraestructura hardware de altas prestaciones. De esta manera, se habilitará la simulación de circuitos cuánticos con un alto número de qubits.

Esta tarea está actualmente en desarrollo activo. Tiene su objetivo de finalización en la anualidad 2022, concretamente en M24. Los primeros trabajos, iniciados en M8, se han concentrado en dos líneas:

- Desarrollo de pruebas de concepto comparativas que permitan obtener primeras impresiones de los frameworks. Estas son de gran valor, ya que suelen existir detalles importantes que son difíciles de detectar solamente a través de la lectura de documentación. Además, se pueden obtener datos de rendimiento que informen la inversión de un mayor esfuerzo en un framework en concreto.
- Diseño de un stack reusable y colaborativo para desarrollo de programas de simulación cuántica. Esta plataforma estará basada en el ecosistema de Jupyter (por ejemplo, JupyterLab, JupyterHub) una herramienta popular en el dominio del análisis de datos para la elaboración de "notebooks" que combinan documentación y código de una manera autocontenida.

Cabe destacar que se está dedicando comparativamente un mayor esfuerzo a los trabajos con PennyLane que con sus alternativas Qiskit y Cirq, tal y como se comenta en las conclusiones de la tarea 4.1. Idealmente, todos los esfuerzos se centrarán en PennyLane a medida que avancen los trabajos, siempre y cuando no se detecten carencias significativas que tengan que ser cubiertas con Qiskit o Cirq.

Justificación 2022

A continuación, se describen las tareas realizadas en este hito de trabajo y los resultados conseguidos.

PT2: Análisis, diseño y desarrollo de algoritmos basados en computación clásica

Este paquete de trabajo se organiza en torno a cuatro tareas, de las cuales dos (T2.1 y T2.2) ya habían sido completadas en la anualidad 2021, una tercera (T2.3), que había sido iniciada en 2021, ha sido finalizada en esta anualidad 2022 y la restante (T2.4) ha sido iniciada y finalizada en esta anualidad.

- T2.1: Análisis del estado de la técnica en algoritmos de computación clásica relacionados con clasificación, optimización y simulación (finalizada en 2021).

- T2.2: Identificación y selección de algoritmos de computación clásica relacionados con clasificación, optimización y simulación (finalizada en 2021).
- T2.3: Diseño e implementación de algoritmos de computación clásica relacionados con clasificación, optimización y simulación (iniciada en 2021 y finalizada en 2022).
- T2.4: Validación de algoritmos de computación clásica relacionados con clasificación, optimización y simulación (iniciada y finalizada en 2022).

A continuación, se describen las acciones desarrolladas en las tareas T2.3 y T2.4 y los principales resultados alcanzados en cada una de ellas.

T2.3: Diseño e implementación de algoritmos de computación clásica relacionados con clasificación, optimización y simulación

Esta anualidad, se ha completado el desarrollo ya iniciado en la anualidad anterior de la tarea T2.3, tal y como refleja el cronograma inicial del proyecto.
Se ha continuado y finalizado el trabajo realizado en lo relativo al diseño e implementación de cada uno de los tres problemas que componen el esquema del proyecto: clasificación, optimización y simulación.
A lo largo de los desarrollos llevados a cabo en el paquete de trabajo "PT2: Análisis, diseño y desarrollo de algoritmos basados en computación clásica", se han abordado los modelos clásicos haciendo uso de conjuntos de datos definidos con mayor complejidad gracias a la mayor madurez tecnológica de dichos algoritmos.

En cuanto al proceso de **clasificación**, el proceso se basa inicialmente en una fase inicial de pre-procesado de los datos para su preparación de cara al modelado posterior. Dicho código puede resumirse en el fragmento mostrado a continuación.

```python
import pandas as pd
import numpy as np
from imblearn.under_sampling import RandomUnderSampler
import warnings
warnings.filterwarnings("ignore")

####################
### PREPROCESADO ###
####################

# Put column names into a list
header_names = list(pd.read_csv('ddos_datasets/03-11/Portmap.csv', nrows=0))

# Remove leading and trailing whitespaces
for i in range(0, len(header_names)):
    header_names[i] = header_names[i].strip()

# List of columns to use
use_columns = ["Flow ID", "Source IP", "Source Port", "Destination IP", "Destination Port",
               "Timestamp", "Flow Duration", "Total Fwd Packets", "Total Backward Packets",
               "Fwd IAT Mean", "Fwd IAT Std", "Fwd IAT Min", "Fwd Packets/s", "Bwd Packets/s",
               "SYN Flag Count", "RST Flag Count", "PSH Flag Count", "ACK Flag Count",
               "URG Flag Count", "Label"]

# USN datasets
dataframe_train_net = pd.read_csv('ddos_datasets/01-12/DrDoS_NetBIOS.csv', names = header_names, skiprows=1, usecols=use_columns)
dataframe_train_net = dataframe_train_net.replace([np.inf, -np.inf], np.nan)
dataframe_train_net = dataframe_train_net.dropna()
dataframe_train_net = dataframe_train_net.loc[:, (dataframe_train_net != 0).any(axis=0)]
dataframe_train_net['Label'] = [1.0 if x == "DrDoS_NetBIOS" else 0.0 for x in dataframe_train_net['Label']]

# Include a list of required features within dataframe_train[]
required_features = ["Flow Duration", "Total Fwd Packets", "Total Backward Packets", "Fwd IAT Mean",
                     "Fwd IAT Std", "Fwd IAT Min", "Fwd Packets/s", "Bwd Packets/s", "RST Flag Count",
                     "ACK Flag Count", "URG Flag Count"]
dataframe_features = dataframe_train_net.loc[:,required_features]

# Create X_train and Y_train variables
Y_train_1 = dataframe_train_net['Label'].values
X_train_1 = dataframe_features.values

# Paso a dataframe
datos = pd.DataFrame(X_train_1, columns = required_features)
datos['Label'] = Y_train_1

# Balanceo seleccionado
under = RandomUnderSampler(sampling_strategy='not minority', random_state=1)
datos_balanced, balanced_labels = under.fit_resample(datos, datos['Label'])
balanced_labels.value_counts()
datos_balanced['Label'] = balanced_labels
```

De este modo, se preparan los datos para poder aplicar la función “modelado” del siguiente modo:

```python
import funciones

funciones.modelado(datos = datos_balanced, semilla = 1, procesadores = 4, cv_folds = 5,
                   ind_RL = True, ind_SVM = False, ind_KNN = True, ind_SGD = True,
                   ind_DT = True, ind_RF = True, ind_AdaBoost = True, ind_GTB = True,
                   ind_NN = True, ind_XGB_tree = True, ind_XGB_linear = True,
                   ind_CatBoost = True, ind_Vot_hard = True, nombre = "DDoS")
```

Ambos fragmentos previos forman parte del script “ddos-classification.py”.
Dicha función “modelado”·, a su vez, ha sido creada y centralizada en el script “funciones.py”, y cuya cabecera se presenta a continuación:

```python
import numpy as np
import pandas as pd
import os
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold, cross_validate, cross_val_score
from openpyxl import load_workbook
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score


def modelado(datos, procesadores = 1, semilla = 1, cv_folds = 10,
             ind_RL = True, ind_SVM = True, ind_KNN = True, ind_SGD = True,
             ind_DT = True, ind_RF = True, ind_AdaBoost = True, ind_GTB = True,
             ind_NN = True, ind_XGB_tree = True, ind_XGB_linear = True,
             ind_CatBoost = True, ind_Vot_hard = True, nombre = ""):
    #
    # Función para el modelado a partir de datos almacenados en dataframe de dos
    # columnas (texto, etiqueta [0,1])
    # la semilla aleatoria fijada y el número de procesadores dado.
    # Permite eliminar modelos que no se deseen con los parámetros de entrada ind_XX
    # pasando de valores por defecto 'True' a valor 'False' en los que se quiera prescindir
    #
```

Con sus distintos parámetros, se selecciona qué modelos aplicar y comparar experimentalmente, pudiendo detectar con respecto a las métricas de interés cuáles son los hiperparámetros asociados a cada modelo que mejores resultados arrojan, así como una comparación global de todos dichos algoritmos, para seleccionar el más adecuado para el caso de uso.
Por último, una vez seleccionado dicho modelo óptimo, se procede con su aplicación y generación con los datos disponibles a lo largo del script "ddos-classification_modelo_final.py". En la siguiente captura, se muestra la fase específica para obtener el mejor resultado para el modelo detectado como óptimo (en este caso, el modelo Gradient Boosting Classifier).

```python
# Control de aleatoriedad
semilla_aleatoria = 1
labels = datos_balanced["Label"]
features = datos_balanced.drop("Label", axis = 1)

from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV

# Grid search
parameters = {'n_estimators': [100, 500],
              'learning_rate': [0.01, 0.1],
              'subsample': [0.6, 1.0],
              'max_depth': [None, 1, 16],
              'max_features': [None, "auto"]}

gbt_grid_search = GridSearchCV(GradientBoostingClassifier(random_state = semilla_aleatoria),
                               parameters, verbose = 10, n_jobs = 4)
gbt_grid_search.fit(features, labels)

# Modelo con parámetros obtenidos
gbt_clf = GradientBoostingClassifier(n_estimators = gbt_grid_search.best_params_["n_estimators"],
                                     learning_rate = gbt_grid_search.best_params_["learning_rate"],
                                     subsample = gbt_grid_search.best_params_["subsample"],
                                     max_depth = gbt_grid_search.best_params_["max_depth"],
                                     max_features = gbt_grid_search.best_params_["max_features"],
                                     random_state = semilla_aleatoria)
gbt_clf.fit(features, labels)
```

Con respecto al proceso de **optimización** de rutas, en primer lugar se han de obtener las distancias y tiempos de viaje relativos a cada par de destinos. Dichas distancias se obtienen haciendo uso de dos herramientas externas: openroute y openstreetmap. El siguiente código resume el proceso de extracción de dicha información.

```python
def distances_times(cnx, nodes):
    """
    Carga de la matriz de distancias y tiempos entre explotaciones.
    :param cnx: conexión a MongoDB.
    :param nodes: identificadores de explotaciones.
    :return: lista con la matriz de tiempos y distancias.
    """
    query = {"codigo": {"$in": nodes}}
    cols = {"_id": 0, "codigo": 1, "lon": 1, "lat": 1}
    cursor = cnx['clientes_geoloc'].find(query, cols)

    df = pd.DataFrame(list(cursor)).drop_duplicates()
    metricas = pd.DataFrame(
        list(cnx.metricas_explotacion.find({"id_o": {"$in": nodes}, "id_d": {"$in": nodes}}))).drop_duplicates()
    try:
        metricas["Metrica"] = -2 * metricas["Metrica"] + 1
        metricas = metricas.sort_values(['id_o', 'id_d'])
        metricas = metricas.pivot(index="id_o", columns="id_d", values="Metrica").fillna(0)
        metricas = metricas.reindex(index=nodes, columns=nodes, fill_value=0)
        metricas = np.array(metricas)

    except:
        metricas = np.zeros((len(nodes), len(nodes)))

    df = df.sort_values('codigo')
    df['lon'] = [str(lon).replace(",", ".") for lon in df['lon']]
    df['lat'] = [str(lat).replace(",", ".") for lat in df['lat']]
    locs = df.loc[:, ["lon", "lat"]]
    locs = [list(locs.loc[i, ['lon', 'lat']]) for i in locs.index]
    try:
        dist_matrix, time_matrix = get_distance_matrix_private(locs)
    except:
        dist_matrix, time_matrix = get_distance_matrix_public(locs)
    dist_matrix = pd.DataFrame(dist_matrix)
    dist_matrix.index = df.codigo
    dist_matrix.columns = df.codigo
    dist_matrix = dist_matrix.fillna(100000)
    dist_matrix = dist_matrix.reindex(index=nodes, columns=nodes, fill_value=100000)

    time_matrix = pd.DataFrame(time_matrix)
    time_matrix.index = df.codigo
    time_matrix.columns = df.codigo
    time_matrix = time_matrix.fillna(100000)
    time_matrix = time_matrix.reindex(index=nodes, columns=nodes, fill_value=100000)

    dist_times = [dist_matrix, time_matrix, metricas]

    return dist_times
```

El proceso de optimización de rutas se realiza mediante un algoritmo evolutivo basado en métodos metaheurísticos y dividido en dos fases. En primer lugar, se realiza una fase de estimación de una solución factible, es decir, que cumpla todas las restricciones del problema. En el siguiente fragmento de código se definen las restricciones del problema a resolver para la obtención de la solución factible inicial. Entre ellas, distancia máxima recorrida para cada vehículo, tiempo máximo en ruta de cada conductor, capacidad máxima de cada vehículo y horarios de apertura de los destinos a visitar.

```python
# Distance constraint
dimension_name = "Distance"
routing.AddDimension(
    transit_callback_index,
    0,  # no slack
    3000,  # Max vehicle travel distance
    True,  # Start cumul to zero
    dimension_name)
distance_dimension = routing.GetDimensionOrDie(dimension_name)

# Capacity constraint
routing.AddDimensionWithVehicleCapacity(
    demand_callback_index,
    0,  # null capacity slack
    vehicle_capacity,  # Vehicle maximum capacities
    True,  # Start cumul to zero
    "Capacity"
)
# Allow to drop nodes
penalty = 100000000
for node in range(0, len(distances)):
    if node != depot:
        if node not in pos_depots:
            routing.AddDisjunction([manager.NodeToIndex(node)], penalty)
        else:
            routing.AddDisjunction([manager.NodeToIndex(node)], 0)

# Time travel constraint
routing.AddDimension(
    transit_time_callback_index,
    0,
    horizon,
    True,
    "Time travel")
time_travel_dimension = routing.GetDimensionOrDie("Time travel")

# Time Window constraint
routing.AddDimension(
    transit_time_callback_index,
    horizon,  # allow waiting time
    horizon,  # maximum time per vehicle
    False,  # Don't force start cumul to zero
    "Time"
)
time_dimension = routing.GetDimensionOrDie("Time")

# Add time window constraints for each location except depot
for location_idx, time_window in enumerate(start_times):
    if location_idx == depot:
        continue
    index = manager.NodeToIndex(location_idx)
    time_dimension.CumulVar(index).SetRange(start_times[location_idx], end_times[location_idx])
```

Posteriormente, se aplica un algoritmo metaheurístico para refinar dicha solución y buscar un óptimo global de acuerdo a una función de coste, en este caso definida como una combinación lineal

entre la distancia recorrida por los vehículos para visitar todos los destinos y la "facilidad" de cada vehículo para recorrer la vía que une dos destinos. En el siguiente fragmento de código se define dicha función de coste.

```
# Cost of each arc
cost_callback_indexes = []
for vehicle in range(len(vehicles)):
    def vehicle_cost_callback(from_index, to_index, i=vehicle):
        from_node = manager.IndexToNode(from_index)
        to_node = manager.IndexToNode(to_index)
        return (1 + alpha * (vehicle_costs[to_node, vehicle] + effective_distances[from_node, to_node]) / 2) * \
               distances[from_node, to_node]

    cost_callback_index = routing.RegisterTransitCallback(vehicle_cost_callback)
    cost_callback_indexes.append(cost_callback_index)
    routing.SetArcCostEvaluatorOfVehicle(cost_callback_index, vehicle)
```

Una vez definidas las restricciones y función de coste del problema, se procede con la ejecución del algoritmo.

```
# Setting first solution heuristic
search_parameters = pywrapcp.DefaultRoutingSearchParameters()

search_parameters.first_solution_strategy = (
    routing_enums_pb2.FirstSolutionStrategy.LOCAL_CHEAPEST_INSERTION)
search_parameters.local_search_metaheuristic = routing_enums_pb2.LocalSearchMetaheuristic.GUIDED_LOCAL_SEARCH


search_parameters.time_limit.seconds = exec_time
search_parameters.solution_limit = sol_limit

# Solve, displays a solution if any.
assignment = routing.SolveWithParameters(search_parameters)

time_end = datetime.now()
dif = (time_end - time_start)

if assignment:
    # out = model_output(manager, routing, assignment, cities, distance_callback, demands,
    #                    time_callback, vehicles, pos_depots)
    #
    # print(out)
    out = model_output_json(manager, routing, assignment, cities, distance_callback, demands,
                            productos, ids_linea_pedido, nums_pedido, vehicles, tipo_granel, pos_depots, origen,
                            df_peticion, df_resume)
    return Response(dumps(out))

return Response(dumps({"msg": "No hay solución"}))
```

Por último, con respecto al problema de **simulación,** los siguientes son los programas que implementan los algoritmos basados en el paradigma clásico para la simulación de moléculas. Hay 4 scripts en Python, los cuales utilizan el módulo psi4 para calcular la energía y el gradiente energético de la molécula.

Dos de estos scripts se aplican a la molécula de agua y otros dos, a la molécula de metano. En ambos casos, se parte de una geometría distorsionada y se utiliza un funcional no-híbrido (BLYP) y un funcional híbrido (B3LYP) para calcular la energía y el gradiente.

Metano con funcional no-híbrido

```
from pickletools import optimize
import psi4
import time

psi4.core.set_output_file("output.dat", False)

start_process = time.process_time()

methane = psi4.geometry(
   """
 C    0.000000    0.000000    0.000000
 H    0.000000    0.000000    1.089000
 H    1.089000    0.000000    0.000000
 H   -0.544500   -0.943102    0.000000
 H   -0.544500    0.943102    0.000000
units angstrom
"""
)
psi4.set_options({"scf_type": "pk", "basis": "3-21G", "reference": "rhf"})

res = {}

# psi4.set_options({'maxiter':'200','d_convergence':'1e-2','e_convergence':'1e-4'})
ene = psi4.energy("scf", dft_functional="blyp", molecule=methane)
ene2, scf_wfn = psi4.optimize(
   "scf", dft_functional="blyp", return_wfn=True, molecule=methane
)
# print(ene,ene2)
res.update({"ps_ene_0": [ene], "ps_ene_f": [ene2]})
HOMO = scf_wfn.epsilon_a_subset("AO", "ALL").np[scf_wfn.nalpha()]
LUMO = scf_wfn.epsilon_a_subset("AO", "ALL").np[scf_wfn.nalpha() + 1]
res["ps_HOMO"] = HOMO
res["ps_LUMO"] = LUMO
res["HOMO-LUMO_gap"] = LUMO - HOMO

end_process = time.process_time()

print(
   "tiempo empleado (usando process_time): {} segundos".format(
      end_process - start_process
   )
)

print(res)
```

Agua con funcional no-híbrido

```
from pickletools import optimize
import psi4
import time

psi4.core.set_output_file("output.dat", False)

start_process = time.process_time()

h2o = psi4.geometry(
   """
O
H 1 0.96
H 1 0.96 2 104.5
"""
)

psi4.set_options({"scf_type": "pk", "basis": "3-21G", "reference": "rhf"})

res = {}

# psi4.set_options({'maxiter':'200','d_convergence':'1e-2','e_convergence':'1e-4'})
ene = psi4.energy("scf", dft_functional="blyp", molecule=h2o)
ene2, scf_wfn = psi4.optimize(
   "scf", dft_functional="blyp", return_wfn=True, molecule=h2o
)
# print(ene,ene2)
res.update({"ps_ene_0": [ene], "ps_ene_f": [ene2]})
HOMO = scf_wfn.epsilon_a_subset("AO", "ALL").np[scf_wfn.nalpha()]
LUMO = scf_wfn.epsilon_a_subset("AO", "ALL").np[scf_wfn.nalpha() + 1]
res["ps_HOMO"] = HOMO
res["ps_LUMO"] = LUMO
res["HOMO-LUMO_gap"] = LUMO - HOMO

end_process = time.process_time()

print(
   "tiempo empleado (usando process_time): {} segundos".format(
      end_process - start_process
   )
)

print(res)
```

Metano con funcional híbrido

```
from pickletools import optimize
import psi4
import time
```

```
psi4.core.set_output_file("output.dat", False)

start_process = time.process_time()

methane = psi4.geometry(
   """
 C    0.000000    0.000000    0.000000
 H    0.000000    0.000000    1.089000
 H    1.089000    0.000000    0.000000
 H   -0.544500   -0.943102    0.000000
 H   -0.544500    0.943102    0.000000
units angstrom
"""
)
psi4.set_options({"scf_type": "pk", "basis": "3-21G", "reference": "rhf"})

res = {}

# psi4.set_options({'maxiter':'200','d_convergence':'1e-2','e_convergence':'1e-4'})
ene = psi4.energy("scf", dft_functional="b3lyp", molecule=methane)
ene2, scf_wfn = psi4.optimize(
   "scf", dft_functional="b3lyp", return_wfn=True, molecule=methane
)
# print(ene,ene2)
res.update({"ps_ene_0": [ene], "ps_ene_f": [ene2]})
HOMO = scf_wfn.epsilon_a_subset("AO", "ALL").np[scf_wfn.nalpha()]
LUMO = scf_wfn.epsilon_a_subset("AO", "ALL").np[scf_wfn.nalpha() + 1]
res["ps_HOMO"] = HOMO
res["ps_LUMO"] = LUMO
res["HOMO-LUMO_gap"] = LUMO - HOMO

end_process = time.process_time()

print(
   "tiempo empleado (usando process_time): {} segundos".format(
      end_process - start_process
   )
)

print(res)
```

Agua con funcional híbrido

```
from pickletools import optimize
import psi4
import time

psi4.core.set_output_file("output.dat", False)

start_process = time.process_time()
```

```
h2o = psi4.geometry(
   """
O
H 1 0.96
H 1 0.96 2 104.5
"""
)

psi4.set_options({"scf_type": "pk", "basis": "3-21G", "reference": "rhf"})

res = {}

# psi4.set_options({'maxiter':'200','d_convergence':'1e-2','e_convergence':'1e-4'})
ene = psi4.energy("scf", dft_functional="b3lyp", molecule=h2o)
ene2, scf_wfn = psi4.optimize(
   "scf", dft_functional="b3lyp", return_wfn=True, molecule=h2o
)
# print(ene,ene2)
res.update({"ps_ene_0": [ene], "ps_ene_f": [ene2]})
HOMO = scf_wfn.epsilon_a_subset("AO", "ALL").np[scf_wfn.nalpha()]
LUMO = scf_wfn.epsilon_a_subset("AO", "ALL").np[scf_wfn.nalpha() + 1]
res["ps_HOMO"] = HOMO
res["ps_LUMO"] = LUMO
res["HOMO-LUMO_gap"] = LUMO - HOMO

end_process = time.process_time()

print(
   "tiempo empleado (usando process_time): {} segundos".format(
      end_process - start_process
   )
)

print(res)
```

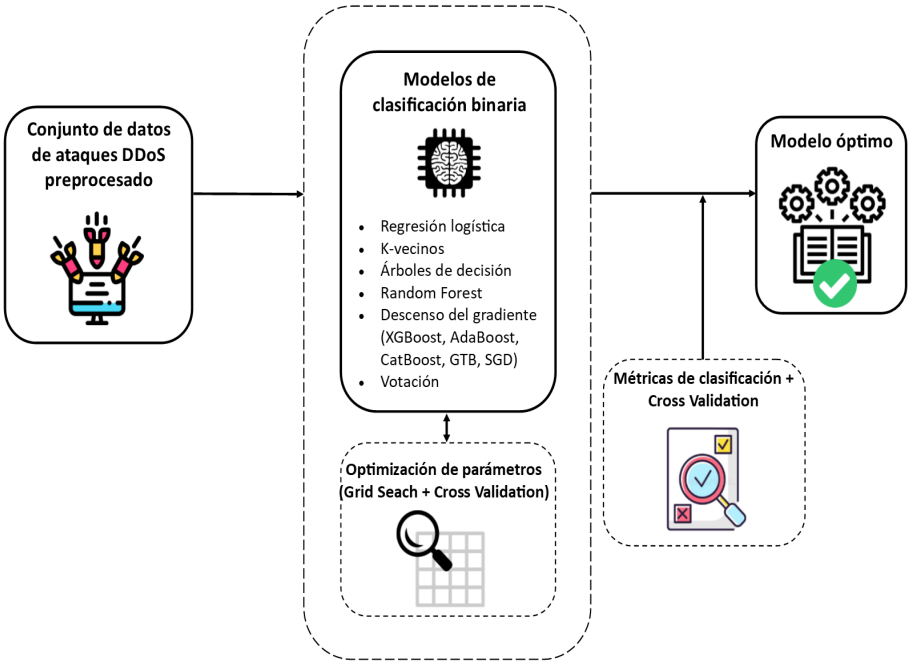
T2.4: Validación de algoritmos de computación clásica relacionados con clasificación, optimización y simulación

En esta tarea se llevó a cabo el estudio de las metodologías finales utilizadas en el proceso de modelado con computación clásica junto a las métricas de validación asociadas.

Clasificación

Ligado a la generación de un sistema para la clasificación de ciberataques en función de sus metadatos. El conjunto de datos utilizado está basado en ataques DDoS (*Distributed denial-of-service*).

El planteamiento del problema se puede ver en el siguiente esquema:



Se listan los distintos modelos de clasificación binaria que se han implementado para el desarrollo del sistema aquí tratado.

- **Regresión logística**
- **K-vecinos**
- **Árboles de decisión**
- ***Random Forest***
- **Métodos del descenso del gradiente**. Dentro de este caso, se han distinguido varias implementaciones diferentes:
    - *XGBoost* (Chen & Guestrin, 2016). Librería de *gradient boosting* distribuido diseñada para ser eficiente, flexible y portátil. Implementa modelos de aprendizaje automático dentro del *framework gradient boosting*.
    - *AdaBoost* (Freund & Schapire, 1997). Modelo estadístico que se puede utilizar junto a otros modelos de aprendizaje para mejorar sus resultados.
    - *CatBoost* (Prokhorenkova, Gusev, Vorobev, Dorogush, & Gulin, 2017). Sistema que permite la adaptación de variables categóricas directamente para el modelado a través del descenso del gradiente.
    - *Gradient Tree Boosting (GTB)* (Friedman, 2001). Se trata de una generalización del *boosting* a funciones de pérdida diferenciables. Se trata de un procedimiento aplicable tanto a problemas de regresión como de clasificación.

- *Stochastic Gradient Descent (SGD)* (Kiefer & Wolfowitz, 1985). Modelo sencillo con el enfoque de ajustar tanto clasificadores como regresores lineales bajo funciones convexas de pérdida, como SVM o regresión logística.

- **Votación** (Ruta & Gabrys, 2005). Sistema para la agregación de modelos de clasificación individuales, los cuales son combinados utilizando un sistema de votación por mayoría para la determinación de la clase final. Este tipo de agregación permite aunar toda la información generada de los diferentes modelos previos, de forma que puedan mitigarse las debilidades individuales de cada uno de ellos, mediante la consideración de los resultados aislados por cada caso.

Parametrización

Tal y como se explica previamente, a cada modelo se le ha asignado diferentes posibles parametrizaciones, las cuales serán contrastadas en cada caso para detectar aquéllas que presenten mejores resultados con respecto a las métricas consideradas. Dichas parametrizaciones son las explicadas a continuación.

Regresión logística (RL)

- **C** (Valor inverso de la fuerza de regularización)
  - Valores considerados: 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000.

K-vecinos (KNN)

- **K** (número de vecinos)
  - Valores considerados: 1, 3, 5, 7, 9, 11, 13, 15
- **Pesos** (función de pesos)
  - Valores considerados: 'uniform', 'distance'
- **Algoritmo** (algoritmo para el cálculo de los vecinos más próximos)
  - Valores considerados: 'ball_tree', 'kd_tree', 'brute'

Árboles de decisión (DT)

- **Criterio** (función de calidad de la división)
  - Valores considerados: 'gini', 'entropy'
- **Splitter** (método para la división en cada nodo)
  - Valores considerados: 'best', 'random'
- **Profundidad máxima del árbol**
  - Valores considerados: None, 1, 2, 4, 8, 16, 32, 64
- **Mínimo de muestras para división**
  - Valores considerados: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0
- **Mínimo de muestras para hoja**
  - Valores considerados: 0.1, 0.2, 0.3, 0.4, 0.5

- **Máximo de variables**
  - Valores considerados: None, ‘sqrt’, ‘log2’

Random Forest (RF)

- **Número de estimadores**
  - Valores considerados: 10, 50, 100, 200
- **Criterio** (función de calidad de la división)
  - Valores considerados: 'gini', 'entropy'
- **Profundidad máxima de los árboles**
  - Valores considerados: None, 1, 4, 16, 32
- **Mínimo de muestras para división**
  - Valores considerados: 0.1, 0.4, 0.7, 1
- **Mínimo de muestras para hoja**
  - Valores considerados: 0.1, 0.3, 0.5
- **Máximo de variables**
  - Valores considerados: None, ‘sqrt’, ‘log2’

Modelos del descenso del gradiente

Stochastic Gradient Descent (SGD)

- **Loss** (función de pérdida)
  - Valores considerados: 'hinge', 'log', 'modified_huber', 'squared_hinge', 'perceptron'
- **Penalti** (término de regularización)
  - Valores considerados: 'l2', 'l1', 'elasticnet'
- **Alfa** (constante que multiplica el término de regularización)
  - Valores considerados: 0.0001, 0.001, 0.01, 0.1, 1
- **Ratio L1** (parámetro asociado únicamente a ‘elasticnet’)
  - Valores considerados: 0, 0.25, 0.5, 0.75, 1

Adaptive Boosting (AdaBoost)

- **Número de estimadores**
  - Valores considerados: 10, 50, 100, 300, 500
- **Ratio de aprendizaje** (peso aplicado a cada clasificador en cada iteración de *boosting)*
  - Valores considerados: 0.001, 0.01, 0.1, 1, 2, 4
- **Algoritmo**
  - Valores considerados: ‘SAMME’, ‘SAMME.R’

Gradient Tree Boosting (GTB)

- **Número de estimadores**
  - Valores considerados: 100, 500

- **Ratio de aprendizaje**
  - Valores considerados: 0.01, 0.1
- **Submuestra** (fracción de la muestra que ha de usarse para el entrenamiento de los *learners* individuales)
  - Valores considerados: 0.6, 1
- **Profundidad máxima**
  - Valores considerados: None, 1, 16
- **Máximo de variables**
  - Valores considerados: None, 'sqrt'

Extreme Gradient Boosting (XGBoost)

Se distinguen en alto nivel dos *boosters* posibles:

- ***tree* (XGB_tree)**.
  - **Ratio de aprendizaje**
    - Valores considerados: 0, 0.5, 1
  - **Profundidad máxima**
    - Valores considerados: 10, 100, 200
- ***linear* (XGB_linear)**.
  - **Lambda** (término de regularización L2 en los pesos)
    - Valores considerados: 1, 3, 5
  - **Alfa** (término de regularización L1 en los pesos)
    - Valores considerados: 0.01, 0.5, 1

CatBoost (CB)

- **Iteraciones**
  - Valores considerados: 100, 200
- **Ratio de aprendizaje**
  - Valores considerados: 0.13, 0.5, 1
- **Profundidad máxima**
  - Valores considerados: 1, 16, 32, 100

Se hará uso de búsqueda exhaustiva de los hiperparámetros de cada uno de los algoritmos considerados, incluyendo cualquier posible combinación de las posibles opciones, y utilizando a su vez validación cruzada con 5 *folds*. Dicho proceso consiste en la división del conjunto de datos disponible en sendos conjuntos equilibrados con respecto a la variable objetivo, y haciendo uso en tantas fases como *folds* de uno de los subconjuntos como test y el resto como entrenamiento. Este proceso permite evitar posibles sesgos en los resultados con respecto a la partición generada.

Estas combinaciones serán contrastadas a través de cuatro métricas distintas, asociadas a problemas de clasificación binaria. En particular, se ha hecho uso de las cuatro siguientes:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F1 - score = \frac{2TP}{2TP + FP + FN}$$

donde TP, TN, FP, FN representan el número de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos, respectivamente.

Con esto, los resultados generados son los que se muestran en la Tabla 1.

Tabla 1. Resultados para el problema de clasificación

Modelo	Accuracy	Precisio n	Recall	F1-score
GTB	0,9985	0,9977	0,9994	0,9985
CatBoost	0,9985	0,9977	0,9994	0,9985
AdaBoost	0,9982	0,9977	0,9988	0,9982
Votación	0,9980	0,9971	0,9988	0,9980
XGB tree	0,9974	0,9959	0,9988	0,9974
KNN	0,9941	0,9919	0,9965	0,9942
DT	0,9438	0,8997	0,9988	0,9467
RF	0,9438	0,8997	0,9988	0,9467

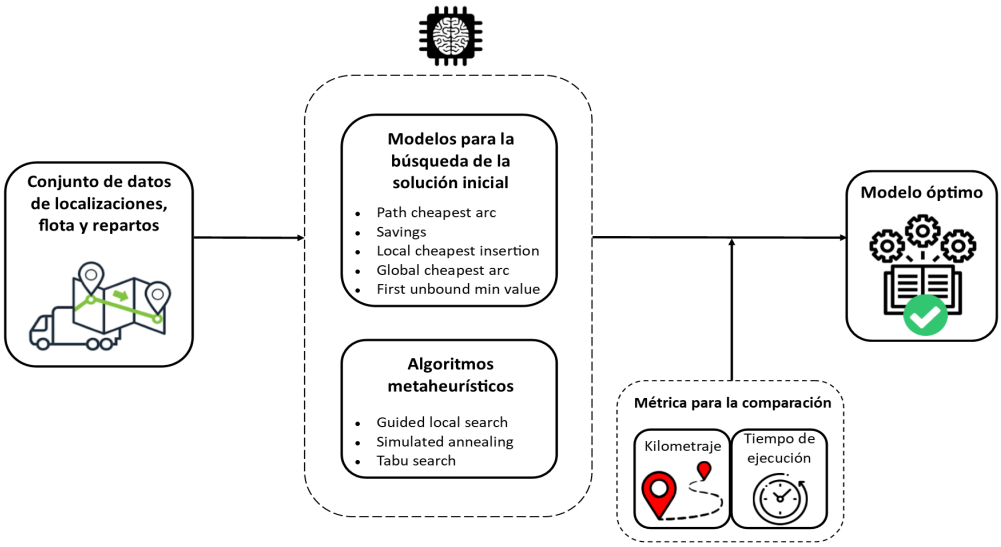
RL	0,9385	0,9059	0,9789	0,9405
SGD	0,9177	0,8843	0,9613	0,9212
XGB linear	0,5006	0,2003	0,4000	0,2669

Tal y como se puede apreciar, todos los modelos presentan valores de *accuracy* superiores al 90%, salvo en el caso del modelo XGBoost con *booster* lineal, el cual presenta malos resultados en general. No solo eso, sino que, para el resto de métricas, salvo en el caso del modelo *Stochastic Gradient Descent* con respecto a la *precision*, todos los valores superan el 0.9.

Por otro lado, se puede ver que existen seis modelos que presentan unos resultados semejantes, donde **todas las métricas superan el 0.99** con las parametrizaciones óptimas, y estos son ***Gradient Tree Boosting*, *CatBoost*, *AdaBoost*, *XGBoost* con *booster* árbol y el método de votación**. Como se puede apreciar, los modelos que mejores resultados presentan son aquellos basados en la metodología del descenso del gradiente, y el de votación que, por definición, está haciendo uso de éstos por ser los que mejores resultados presentan.

Optimización

Se aplica a un caso de uso de optimización de rutas. El planteamiento del problema se describe en el siguiente esquema:



Se tendrán en cuenta dos tipos de modelos diferentes para la generación del sistema de optimización de rutas de este proyecto: los modelos para la búsqueda de solución inicial y los algoritmos metaheurísticos.

Búsqueda de la solución inicial

Path cheapest arc

El primer método seleccionado tiene como base la elección como punto de partida de la ruta a un nodo, de tal forma que se conecte a través de un arco con el nodo que asocie un menor coste. Se trata de un proceso iterativo que se lleva a cabo hasta que se hayan recorrido todos los nodos disponibles y se haya vuelto al punto de partida.

Savings

Se trata de un algoritmo de optimización local, el cual se basa en la generación de ahorros o *savings* a la hora de la asignación de las rutas. Se trata de un proceso iterativo:

1. Generación de tantas rutas distintas, como nodos compongan el problema.
2. Generación de los ahorros o *savings* obtenidos tras la unificación de dos nodos a una ruta única.
3. Ordenación de dichos ahorros o *savings* decrecientemente.
4. Selección del mayor ahorro o *saving* y unificación de tales nodos en una sola ruta, siempre y cuando se sigan respetando las restricciones generales, y mantenga coherencia.
5. Repetición de paso 2 en caso de haber nodos sin alcanzar.
6. Repetición de pasos 3 y 4 hasta que no haya cambios en los ahorros o *savings*.

Local cheapest insertion

Método para la creación de forma iterativa de una posible solución a través de la inserción de los nodos en la posición más "barata". Dicho coste, viene dado por la función de coste de arco. En este caso, los nodos son insertados por orden de creación, por lo que el algoritmo es más ágil computacionalmente.

Global cheapest arc

Método basado en la conexión de los nodos que generan un arco con el menor coste asociado posible. No se trata de una inserción necesariamente iterativa, sino que dichos arcos podrán ser incorporados de forma independiente.

First unbound min value

Esta opción busca seleccionar como primer nodo a un sucesor no vinculado, y lo conecta posteriormente al primer nodo que esté disponible.

Metaheurísticos

Tras haber seleccionado la primera solución con alguna de las opciones presentadas previamente, el sistema de optimización llevará a cabo diferentes variaciones sobre ésta, buscando así el óptimo global del problema. Para tal fin, se hace uso de los conocidos como algoritmos metaheurísticos.

Este tipo de modelos tiene diferentes aspectos relevantes a prestar atención, siendo el principal la capacidad para la divergencia, es decir, que el algoritmo no es capaz de volver a encontrar soluciones óptimas, o bien que el tiempo necesario para hacerlo sea demasiado alto como para resultar viable en el ámbito concreto de aplicación.

Dicho problema se intenta abordar para garantizar la obtención de soluciones útiles en el tiempo, añadiendo restricciones tanto en cuestión temporal como en la cantidad de iteraciones asociadas.

Al igual que con la búsqueda de solución inicial, se contemplarán diferentes opciones de algoritmos metaheurísticos para este problema, que se pasan a describir brevemente a continuación.

Guided local search

Método basado en el uso de penalizaciones ficticias con respecto a la función objetivo, de tal modo que sea posible evitar óptimos locales en situaciones de estancamiento, siendo posible así identificar óptimos globales. Así, dicho algoritmo buscará una solución hasta su estancamiento en un óptimo local, tras lo cual, se procede a penalizar las características propias de tal punto, consiguiendo salir de dicho estancamiento. Este proceso es repetido durante el número de iteraciones fijado.

Simulated annealing

Este segundo método tiene como base la consideración de una función de temperatura, la cual decrece progresivamente, y que busca un candidato a nuevo óptimo en la zona cercana en el espacio

del óptimo actual. Tras dicha selección, se realiza la comprobación para determinar si dicho valor es mejor o no que el previo y, en caso afirmativo, se conserva dicho punto como óptimo actual. En caso negativo, se analizará la diferencia entre dicho nuevo punto y la temperatura actual, pudiendo así no ser descargado si dicho valor no empeora de forma notoria el actual. Así, se evitarán estancamientos en óptimos locales.

Tabu search

El método *tabu search* selecciona el mejor candidato entre los disponibles como vecinos del óptimo actual, aun siendo éste peor que el anterior. También se mantiene una lista que permita vetar elecciones concretas a corto plazo, si ya se han realizado previamente. Dicha lista se conoce como lista tabú, y proporciona la capacidad para salir de óptimos locales.

Los resultados obtenidos para la combinación de los cinco modelos de búsqueda de solución local, y los tres algoritmos metaheurísticos considerados, se presentan en la siguiente Tabla con respecto al kilometraje medio y al tiempo de ejecución.

Tabla. Resultados para el problema de optimización

Solución local	Metaheurístico	Kilometraje	Tiempo de ejecución
Path cheapest arc	***Guided local search***	2204.8	900.4622
Path cheapest arc	***Simulated annealing***	2236.0	900.4636
Path cheapest arc	***Tabu search***	2236.0	900.4775
Savings	***Guided local search***	2223.4	900.4460
Savings	***Simulated annealing***	2239.0	900.4612
Savings	***Tabu search***	2239.0	900.4547

Local cheapest insertion	***Guided local search***	2172.0	900.4587
Local cheapest insertion	***Simulated annealing***	2248.8	900.4652
Local cheapest insertion	***Tabu search***	2246.8	900.4634
Global cheapest arc	***Guided local search***	2225.6	900.4627
Global cheapest arc	***Simulated annealing***	2210.8	900.4608
Global cheapest arc	***Tabu search***	2210.8	900.4602
First unbound min value	***Guided local search***	2234.4	900.4530
First unbound min value	***Simulated annealing***	2284.4	900.6426
First unbound min value	***Tabu search***	2284.4	900.6096

Analizando los resultados presentados en la Figura 2, se puede identificar que los tiempos de ejecución de todas las combinaciones son prácticamente idénticos, por lo que carece de sentido tomar decisiones basadas en dicho parámetro. Así, atendiendo al kilometraje medio, se puede observar como la combinación de algoritmos con mejor resultado se corresponde al algoritmo de búsqueda de solución local *Local cheapest insertion* junto con el algoritmo metaheurístico *Guided local search*, teniendo como valor de kilometraje medio asociado 2.172 km, siendo la única combinación por debajo de los 2.200 km medios.

Simulación

Uno de los dominios de aplicabilidad donde la computación cuántica está llamada a marcar la diferencia dentro del mundo de la investigación es el ámbito de la química computacional, en concreto las simulaciones atomísticas siguiendo los postulados de la mecánica cuántica.

Métodos seleccionados

DFT

Dado que la función de onda para un sistema de N electrones es una función de 3N variables espaciales independientes, resolver la ecuación electrónica de Schrödinger utilizando enfoques tradicionales de química cuántica ab-initio, como la interacción de configuración (CI), se vuelve computacionalmente muy exigente para grandes complejos moleculares.

La teoría del funcional de la densidad (DFT por sus siglas en inglés) ofrece una alternativa valiosa que es computacionalmente más eficiente y, además, bastante precisa. Esta teoría se basa en el teorema de Hohenberg-Kohn [1] que muestra que es posible usar solo la densidad electrónica para calcular la energía del estado fundamental E_0.

De ahí que la complejidad del problema disminuya drásticamente, ya que la densidad electrónica es función de solo tres coordenadas, y es independiente del número de electrones. Dado un potencial externo $v_{\text{ext}}(\mathbf{r})$, la energía electrónica total se puede expresar como un funcional de la densidad electrónica $\rho(\mathbf{r})$,

$$E \equiv E_v\left[\rho(\mathbf{r})\right] \equiv \int v_{\text{ext}}(\mathbf{r})\rho(\mathbf{r})d\mathbf{r} + F\left[\rho(\mathbf{r})\right]$$

donde $F[\rho(\mathbf{r})]$ es un funcional universal y la integral recorre todo el espacio. Para los sistemas moleculares, el potencial externo $v_{ext}(\mathbf{r})$ suele ser el potencial generado por los núcleos (ver ecuación 2.9).

$$\hat{V}_{\text{e-N}} = -\frac{1}{4\pi\varepsilon_0}\sum_i^n\sum_I^N\frac{Z_I e^2}{r_{iI}}$$

El teorema de Hohenberg-Kohn también demuestra que se cumple un principio variacional para el funcional de energía: para cualquier densidad, la energía dada por el funcional de energía correspondiente nunca es más pequeña que la energía del estado fundamental, y la energía del estado fundamental se obtiene usando solamente la densidad electrónica exacta del estado fundamental.

La energía total del funcional se puede escribir como una suma de diferentes términos:

$$E_v\left[\rho\right] = T\left[\rho\right] + V_{\text{e-N}}\left[\rho\right] + V_{\text{e-e}}\left[\rho\right]$$

donde $T[\rho]$ es la energía cinética de los electrones, $V_{e\text{-}N}[\rho]$ es la energía debida a la atracción electrón-núcleo y $V_{e\text{-}e}[\rho]$ es la energía de repulsión electrón-electrón, que se puede descomponer en la contribución Coulombica clásica y las contribuciones no clásicas:

$$V_{e\text{-}e}[\rho] = J[\rho] + V_{e\text{-}e}^{nc}[\rho]$$

Mientras que $V_{e\text{-}N}[r]$ y $J[\rho]$ pueden ser calculados en términos de la densidad electrónica como

$$V_{e\text{-}N}[\rho] = \int v_{ext}(\mathbf{r})\rho(\mathbf{r})d\mathbf{r}$$

y

$$J[\rho] = \int \frac{\rho(\mathbf{r})\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}d\mathbf{r}'$$

la forma explícita del funcional de la energía cinética y de la de energía de repulsión electrón-electrón no clásica no es conocida. Con el fin de simplificar los cálculos del término de la energía cinética, Kohn y Sham propusieron calcular $T_s[\rho]$, que es la energía cinética de un conjunto de partículas independientes. Este término se puede escribir fácilmente en términos de un conjunto de orbitales, conocidos como orbitales de Kohn-Sham. La parte restante de la energía cinética es la energía de correlación cinética $T_c[\rho]$, lo cual conduce a

$$T[\rho] = T_c[\rho] + T_s[\rho]$$

La contribución $T_c[\rho]$ de la energía cinética se combina luego con el término no clásico de repulsión electrón-electrón en el funcional de intercambio-correlación,

$$E_{xc}[\rho] = T_c[\rho] + V_{ee}^{nc}[\rho],$$

y el funcional energético puede ser reescrito como

$$E_v[\rho] = T_s[\rho] + V_{e\text{-}N}[\rho] + J[\rho] + E_{xc}[\rho]$$

Funcional de intercambio-correlación

Se desconoce la forma exacta del funcional de intercambio-correlación y, por lo tanto, la principal aproximación en DFT está en el tratamiento de este. Una de las primeras aproximaciones propuestas es la aproximación de densidad local (LDA por sus siglas en inglés) en la que el sistema se trata localmente como un gas de electrones uniforme utilizando

$$E_{xc}^{LDA}[\rho] = \int e_{xc}(\rho(\mathbf{r}))\rho(\mathbf{r})d\mathbf{r}$$

donde la función de energía de intercambio-correlación ha sido calculada con mucha exactitud a través de simulaciones cuánticas de Monte Carlo para el gas de electrones homogéneos. Está aproximación (LDA) ha resultado funcionar bastante bien para metales y semiconductores, sin embargo, funciona mal para moléculas, especialmente en la predicción de energías de enlace. Esto no es de extrañar ya que la densidad electrónica en los sistemas moleculares es muy heterogénea. Para

mejorar el funcional de intercambio-correlación, se usa la aproximación de gradiente generalizado (GGA), incluyendo el gradiente de la densidad:

$$E_{xc}^{GGA}[\rho,\nabla\rho]=\int f(\rho(\mathbf{r}),\nabla\rho(\mathbf{r}))d\mathbf{r}$$

.

El funcional GGA que se va a utilizar es BLYP, que combina el funcional de intercambio de Becke con el funcional de correlación de Lee, Yang y Parr.

Otros funcionales que se utilizan ampliamente son los funcionales híbridos. En estos se mezcla parte de la energía exacta de intercambio de Hartree-Fock con el funcional GGA, siguiendo la siguiente fórmula:

$$E_{xc}^{hybrid}[\rho]=E_{xc}^{GGA}[\rho,\nabla\rho]+c_x\left(E_x^{HF}[\rho]-E_x^{GGA}[\rho]\right)$$

El parámetro c_x determina la mezcla entre la parte Hartree-Fock y la GGA. El funcional híbrido que se va a utilizar es B3LYP, donde la cantidad de intercambio exacto es del 20%.

Modelos seleccionados

Los modelos seleccionados inicialmente son la molécula de agua y la de metano.

En ambos casos la geometría suministrada de partida ha sido distorsionada para así poder valorar de forma más sencilla si la optimización geométrica se desarrolla correctamente.

La geometría de las moléculas seleccionadas se suministra mediante coordenadas absolutas, es decir, hay una línea por átomo y para cada uno de ellos se suministran tres valores, uno por coordenada espacial (x, y, z).

```
O
H 1 0.96
H 1 0.96 2 104.5
```

Molécula de agua

```
C     0.000000     0.000000     0.000000
H     0.000000     0.000000     1.089000
H     1.089000     0.000000     0.000000
H    -0.544500    -0.943102     0.000000
H    -0.544500     0.943102     0.000000
```

Molécula de metano

Para cada uno de los compuestos seleccionados se ha llevado a cabo un proceso de optimización geométrica para minimizar la energía total del sistema. Esta modelización consiste, en un primer

paso, en el cálculo de la energía del estado fundamental de la molécula en la geometría inicial. A partir de ahí, ligeras modificaciones sistemáticas de la geometría de la molécula son llevadas a cabo junto con el cálculo de la energía del estado fundamental para cada nuevo juego de coordenadas, hasta encontrar la geometría asociada al mínimo energético.

Asimismo, una vez optimizada la geometría también se ha llevado a cabo un cálculo energético de los orbitales moleculares para obtener la energía del último orbital ocupado (HOMO por sus siglas en inglés) y del primer orbital no ocupado (LUMO por sus siglas en inglés), así como la diferencia energética de estos. Estos valores suelen ser bastante importantes, ya que están fuertemente relacionados con una serie de propiedades, tales como la energía de ionización, la afinidad electrónica y los espectros de absorción y emisión.

Parametrización

Tanto para la molécula de agua como para la de metano, las simulaciones se van a realizar dos veces, una usando un funcional no-híbrido y otra usando uno híbrido, de forma que podamos desarrollar y evaluar dentro del ámbito de la computación cuántica un mayor abanico de opciones. El funcional no-híbrido seleccionado es BLYP, mientras que el híbrido es B3LYP, como se explica en la sección "Métodos seleccionados".

Por otro lado, la simulación se llevará a cabo aplicando el basis-set 3-21G. Un basis-set en química teórica y computacional es un conjunto de funciones (llamadas funciones básicas) que se utiliza para representar la función de onda electrónica, y así convertir las ecuaciones diferenciales parciales del modelo en ecuaciones algebraicas adecuadas para una implementación computacional eficiente.

Otra de las opciones a tener en cuenta en las simulaciones es que estas se llevan a cabo usando cálculos Hartree-Fock restringidos (RHF por sus siglas en inglés), lo cual implica que simplificamos el modelo forzando a que todos los electrones estén emparejados y que la molécula se encuentre siempre en su estado singlete (multiplicidad de spin = 1). Más adelante se podrían plantear casos usando ecuaciones Hartree-Fock no restringidas y/o ecuaciones Hartree-Fock restringidas de capa abierta (UHF o ROHF de sus siglas en inglés), para así poder simular compuestos con electrones desemparejados, como por ejemplo la molécula de oxígeno.

Resultados

A través de la computación clásica y usando la librería Psi4[5] de Python, se han llevado a cabo las simulaciones de las moléculas propuestas a partir de geometrías distorsionadas. En ambos casos, se ha utilizado un *script* con un funcional no híbrido (BLYP) y otro con un funcional híbrido (B3LYP). El proceso a seguir es:

1. Calcular la energía para la geometría a través de la función de onda.
2. Modificar la geometría de la molécula.
3. Calcular la energía mediante la función de onda para la nueva geometría.

4. Mediante el gradiente energético, decidimos si estamos optimizando la geometría (buscamos el mínimo energético).
5. Si la energía ha disminuido volvemos al punto 2); si la energía no ha disminuido deshacemos el cambio y modificamos la geometría en otra dirección.
6. Repetir hasta alcanzar el mínimo energético.

Los resultados han sido satisfactorios para las simulaciones para los casos de estudios propuestos, pudiendo comprobar que en el caso de la optimización geométrica partiendo de geometrías distorsionadas, tanto el metano como el agua acaban convergiendo en las geometrías esperadas. En cuanto al esfuerzo computacional para llevar a cabo estas simulaciones, hemos de tener en cuenta que hablamos de moléculas muy simples y de métodos y software muy bien asentados y optimizados, por lo que hablamos de cálculos que se llevan a cabo en órdenes de tiempo de segundos, o menores.

PT3: Análisis, diseño y desarrollo de algoritmos basados en computación cuántica

Este paquete de trabajo se organiza en torno a cuatro tareas, de las cuales una (T3.1) había sido finalizada en su totalidad en la anualidad 2021. Una segunda (T3.2) había sido iniciada en la anualidad 2021 y se finalizó en esta anualidad 2022. Las dos restantes (T2.3 y T2.4) se iniciaron y finalizaron a lo largo de esta anualidad.

- T3.1: Análisis del estado de la técnica en algoritmos de computación cuántica relacionados con clasificación, optimización y simulación (finalizada en 2021).
- T3.2: Identificación y selección de algoritmos de computación cuántica relacionados con clasificación, optimización y simulación (iniciada en 2021 y finalizada en 2022).
- T3.3: Diseño e implementación de algoritmos de computación cuántica relacionados con clasificación, optimización y simulación (iniciada y finalizada en 2022).
- T3.4: Validación de algoritmos de computación cuántica relacionados con clasificación, optimización y simulación (iniciada y finalizada en 2022).

A continuación, se describen las acciones desarrolladas en las tareas T3.2, T3.3 y T3.4 y los principales resultados alcanzados.
A lo largo de los desarrollos llevados a cabo en el paquete de trabajo "PT3: Análisis, diseño y desarrollo de algoritmos basados en computación cuántica", se ha procedido inicialmente con casos de uso más sencillos que en el caso de la computación clásica, ya que se pasará a los conjuntos abordados en el contexto clásico a lo largo de los desarrollos del "PT5: Demostración de casos de uso".

T3.2: Identificación y selección de algoritmos de computación cuántica relacionados con clasificación, optimización y simulación

La mayor parte del trabajo en esta tarea 3.2 se realizó en la anualidad 2021 y se concluyó en el M13. La tarea tiene como objetivo la revisión de los algoritmos identificados en la tarea 3.1, de manera que se pueda seleccionar un conjunto definitivo que pasará a la fase de implementación. Para ello, se han considerado principalmente la viabilidad de dicha implementación y los recursos disponibles en el ecosistema.

Las conclusiones se describen en los siguientes párrafos.

En el dominio de la **clasificación**, la decisión es más compleja. Existe un número mucho más elevado de algoritmos. Además, no se han encontrado referencias que proporcionen datos de rendimiento comparativos con un nivel de exhaustividad apropiado. Inicialmente, las técnicas de Quantum Transfer Learning y Data Re-Uploading Universal Classifier apuntan a ser las más interesantes por su viabilidad para ser implementadas en simuladores y ordenadores cuánticos NISQ razonablemente accesibles. Aun así, cerrar las puertas a otras técnicas, como por ejemplo las redes Quanvolutional, tendría demasiado riesgo. Es por esto que se recomienda no descartar ninguna técnica de antemano.

De manera similar al caso anterior, en el dominio de la **optimización** de rutas el QAOA (Quantum Approximate Optimization Algorithm) y el VQE (Variational Quantum Eigensolver) son los algoritmos cuánticos que más se aplican en la actualidad. Entre ellos, el VQE tiene la ventaja de utilizar técnicas de optimización clásicas en combinación con los circuitos cuánticos. Esto permite una mejor exploración del espacio de resultados posibles, lo que aumenta la probabilidad de encontrar soluciones óptimas o cercanas a las óptimas. En comparación, el QAOA utiliza una estrategia de búsqueda más limitada, lo que puede resultar en soluciones subóptimas.
En lo que al escalado se refiere, el VQE puede aprovechar mejor la capacidad de los circuitos cuánticos para manejar grandes conjuntos de datos, lo que lo convierte en una opción más atractiva para problemas de logística de gran escala. Cabe resaltar que en el E3.2 entramos en más detalle sobre cómo funciona cada uno de ellos.

En el dominio de la **simulación química**, el algoritmo Variational Quantum Eigensolver (VQE) es el candidato más adecuado. El objetivo fundamental del algoritmo VQE es la determinación del estado de mínima energía de un hamiltoniano dado. Este problema, fundamental en física y química cuánticas, presenta multitud de aplicaciones en el estudio de sistemas cuánticos. Puesto que el número de parámetros que describen el estado de uno de estos sistemas crece exponencialmente con el número de partículas del mismo, se considera altamente improbable que se puedan encontrar algoritmos clásicos que resuelvan este tipo de tareas eficientemente.
VQE se propone, precisamente, como un algoritmo híbrido clásico-cuántico que utiliza técnicas de computación cuántica para acelerar aquellas partes del proceso de determinación del estado de mínima energía de un sistema que resultan costosas con ordenadores tradicionales. En concreto, el ordenador cuántico interviene en el proceso de determinación de la energía de un cierto estado, delegando en ordenadores clásicos el resto de fases del algoritmo.

T3.3: Diseño e implementación de algoritmos de computación cuántica relacionados con clasificación, optimización y simulación

A continuación, se muestran diferentes capturas del software utilizado para la resolución de los diferentes subproblemas matemáticos mediante computación cuántica.

Clasificación
Modelo variacional cuántico:

```python
@staticmethod
#Funcion que añade un bias al resultado del circuito cuantico
def quantum_model_plus_bias(device,x,params, bias):
    @qml.qnode(device, interface="torch", diff_method="parameter-shift")
    def quantum_model(x, params):
        #AngleEmbedding para asociar valores del dataset a ángulos de un qubit
        AngleEmbedding(x, wires=range(len(device.wires)))
        #StronglyEntanglingLayers para realizar un entrelazamiento fuerte entre
        #todos los qubits del circuito
        StronglyEntanglingLayers(params, wires=range(len(device.wires)))
        #Medición de la posición del qubit en el eje Z
        return qml.expval(qml.PauliZ(0))
    return quantum_model(x,params) + bias


#Función de pérdida para medir utilizar en la funcion de coste
@staticmethod
def hinge_loss(predictions, targets):
    """Implements the hinge loss."""
    all_ones = torch.ones_like(targets)
    hinge_loss = all_ones - predictions * targets
    #Dado que la función max(0,x) es no diferenciable, utilizamos
    #su equivalente matemático, relu, en su lugar
    hinge_loss = relu(hinge_loss)
    return hinge_loss
```

Clasificador K-vecinos cuántico

```python
#Creamos el device sobre el que montar el circuito
dev = qml.device("default.qubit", wires = 5)

#Función de mapeo de los datos de entrada a qubits
def feature_map(x, wires):
    qml.RY(x[0], wires = wires[0])
    qml.RY(x[1], wires = wires[1])

#Circuito cuántico
@qml.qnode(dev)
def swap_test(x1,x2):

    #Mapeamos los datos
    feature_map(x1, wires = [1,2])
    feature_map(x2, wires = [3,4])

    #Aplicamos operaciones sobre los mismos
    qml.Hadamard(wires = 0)
    qml.CSWAP(wires = [0,1,3])
    qml.CSWAP(wires = [0,2,4])
    qml.Hadamard(wires = 0)

    return qml.expval(qml.PauliZ(0))

#Función que calcula la distancia entre 2 vecinos utilizando el circuito cuántico anterior
def distance(x1, x2):
    return 2 - 2 * swap_test(x1,x2)
```

Modelo basado en kernels

```python
#Definimos el kernel, es decir, el ciruito
@qml.qnode(dev_kernel,diff_method="parameter-shift")
def kernel(x1, x2):
    #Se usa la plantilla de AngleEmbedding para asignar los datos a ángulos de la esfera
    AngleEmbedding(x1, wires=range(n_qubits))
    qml.adjoint(AngleEmbedding)(x2, wires=range(n_qubits))
    return qml.expval(qml.Hermitian(projector, wires=range(n_qubits)))

#Función que cosntruye una matriz cuyas entradas son los datos evaluados por
#el circuito de 2 en 2
def kernel_matrix(A, B):
    return np.array([[kernel(a, b) for b in B] for a in A])
```

Optimización

Variational Quantum Eigensolver (VQE)

```python
class QuantumOptimizer:
    def __init__(self, instance, n, K):

        self.instance = instance
        self.n = n
        self.K = K

    def binary_representation(self, x_sol=0):

        instance = self.instance
        n = self.n
        K = self.K

        A = np.max(instance) * 100  # A parameter of cost function

        # Determine the weights w
        instance_vec = instance.reshape(n**2)
        w_list = [instance_vec[x] for x in range(n**2) if instance_vec[x] > 0]
        w = np.zeros(n * (n - 1))
        for ii in range(len(w_list)):
            w[ii] = w_list[ii]

        # Some variables I will use
        Id_n = np.eye(n)
        Im_n_1 = np.ones([n - 1, n - 1])
        Iv_n_1 = np.ones(n)
        Iv_n_1[0] = 0
        Iv_n = np.ones(n - 1)
        neg_Iv_n_1 = np.ones(n) - Iv_n_1
```

Simulación

VQE (Variational Quantum Eigensolver)

Implementación basada en **Qiskit** de la determinación de la **energía del *ground state*** de una molécula de dihidrógeno (H_2) así como de su **perfil de disociación**.

+*In[ ]:*+
[source, ipython3]
---
pip install qiskit==0.39.2

+*In[ ]:*+
[source, ipython3]
---
pip install qiskit_nature==0.4.5

+*In[ ]:*+
[source, ipython3]
---
pip install pyscf==2.1.1

```
----


+*In[ ]:*+
[source, ipython3]
----
from qiskit_nature.drivers import Molecule
from qiskit_nature.drivers.second_quantization import ElectronicStructureMoleculeDriver,
ElectronicStructureDriverType
from qiskit_nature.problems.second_quantization import ElectronicStructureProblem

mol = Molecule(geometry=[['H', [0., 0., -0.37]],
                 ['H', [0., 0., 0.37]]])
driver = ElectronicStructureMoleculeDriver(mol, basis='sto3g',
      driver_type=ElectronicStructureDriverType.PYSCF)
problem = ElectronicStructureProblem(driver)
secqop = problem.second_q_ops()
print(secqop[0])
----


+*In[ ]:*+
[source, ipython3]
----
from qiskit_nature.converters.second_quantization import QubitConverter
from qiskit_nature.mappers.second_quantization import JordanWignerMapper

qconverter = QubitConverter(JordanWignerMapper())
qhamiltonian = qconverter.convert(secqop[0])
print("Qubit Hamiltonian")
print(qhamiltonian)
----


+*In[ ]:*+
[source, ipython3]
----
pip install pylatexenc==2.10
----


+*In[ ]:*+
[source, ipython3]
----
from qiskit.circuit.library import EfficientSU2

ansatz = EfficientSU2(num_qubits=4, reps=1, entanglement="linear",
   insert_barriers = True)
ansatz.decompose().draw("mpl")
----


+*In[ ]:*+
[source, ipython3]
----
from qiskit.algorithms import VQE
from qiskit import Aer
```

```
from qiskit.utils import QuantumInstance, algorithm_globals
import numpy as np
from qiskit.algorithms.optimizers import COBYLA

seed = 1234
np.random.seed(seed)
algorithm_globals.random_seed = seed

optimizer = COBYLA()

initial_point = np.random.random(ansatz.num_parameters)
quantum_instance = QuantumInstance(backend =
   Aer.get_backend('aer_simulator_statevector'))

vqe = VQE(ansatz=ansatz, optimizer=optimizer,
   initial_point=initial_point,
   quantum_instance=quantum_instance)
----


+*In[ ]:*+
[source, ipython3]
----
result = vqe.compute_minimum_eigenvalue(qhamiltonian)
print(result)
----


+*In[ ]:*+
[source, ipython3]
----
from qiskit.algorithms.minimum_eigensolvers import NumPyMinimumEigensolver
solver = NumPyMinimumEigensolver()
result = solver.compute_minimum_eigenvalue(qhamiltonian)
print(result)
----


+*In[ ]:*+
[source, ipython3]
----
from qiskit.algorithms import VQD
vqd = VQD(ansatz=ansatz,
   optimizer=optimizer,
   initial_point=initial_point,
   quantum_instance=quantum_instance,
   k = 2)
result = vqd.compute_eigenvalues(qhamiltonian)
print(result)
----


+*In[ ]:*+
[source, ipython3]
----
from qiskit_nature.algorithms import GroundStateEigensolver
```

```
solver = GroundStateEigensolver(qconverter, vqe)
result = solver.solve(problem)
print(result)
----


+*In[ ]:*+
[source, ipython3]
----
from qiskit_nature.algorithms import VQEUCCFactory

vqeuccf = VQEUCCFactory(quantum_instance = quantum_instance)
----


+*In[ ]:*+
[source, ipython3]
----
vqeuccf.get_solver(problem, qconverter).ansatz.decompose().draw("mpl")

----


+*In[ ]:*+
[source, ipython3]
----
solver = GroundStateEigensolver(qconverter, vqeuccf)
result = solver.solve(problem)
print(result)
----


+*In[ ]:*+
[source, ipython3]
----
import matplotlib.pyplot as plt

energies = []
solver = GroundStateEigensolver(qconverter, vqe)
distances = np.arange(0.2, 2.01, 0.01)
for d in distances:
  mol = Molecule(geometry=[['H', [0., 0., -d/2]],
                           ['H', [0., 0., d/2]]])

  driver = ElectronicStructureMoleculeDriver(mol, basis='sto3g',
      driver_type=ElectronicStructureDriverType.PYSCF)
  problem = ElectronicStructureProblem(driver)
  result = solver.solve(problem)
  energies.append(result.total_energies)

plt.plot(distances, energies)
plt.title('Dissociation profile')
plt.xlabel('Distance')
plt.ylabel('Energy')
plt.savefig('esu2.pdf')
----
```

```
+*In[ ]:*+
[source, ipython3]
----
import matplotlib.pyplot as plt

energies = []
solver = GroundStateEigensolver(qconverter, vqeuccf)
distances = np.arange(0.2, 2.01, 0.01)
for d in distances:
  mol = Molecule(geometry=[['H', [0., 0., -d/2]],
                           ['H', [0., 0., d/2]]])

  driver = ElectronicStructureMoleculeDriver(mol, basis='sto3g',
      driver_type=ElectronicStructureDriverType.PYSCF)
  problem = ElectronicStructureProblem(driver)
  result = solver.solve(problem)
  energies.append(result.total_energies)

plt.plot(distances, energies)
plt.title('Dissociation profile')
plt.xlabel('Distance')
plt.ylabel('Energy')
plt.savefig('uccsd.pdf')
----
```

T3.4: Validación de algoritmos de computación cuántica relacionados con clasificación, optimización y simulación

En este documento se presenta el proceso llevado a cabo para implementar y validar los algoritmos de clasificación propuestos para generar un modelo de detección de anomalías. Asimismo, se presenta la validación para los algoritmos de optimización y simulación enmarcados dentro de los retos de optimización de procesos industriales y diseño de nuevos productos, fármacos específicamente.

Clasificación

Para los 3 modelos anteriores se ha utilizado el mismo conjunto de datos. Este consiste en un subconjunto del conjunto de datos Iris escalado ya que este proporciona un balance entre un número de variables de entrenamiento y número de elementos.

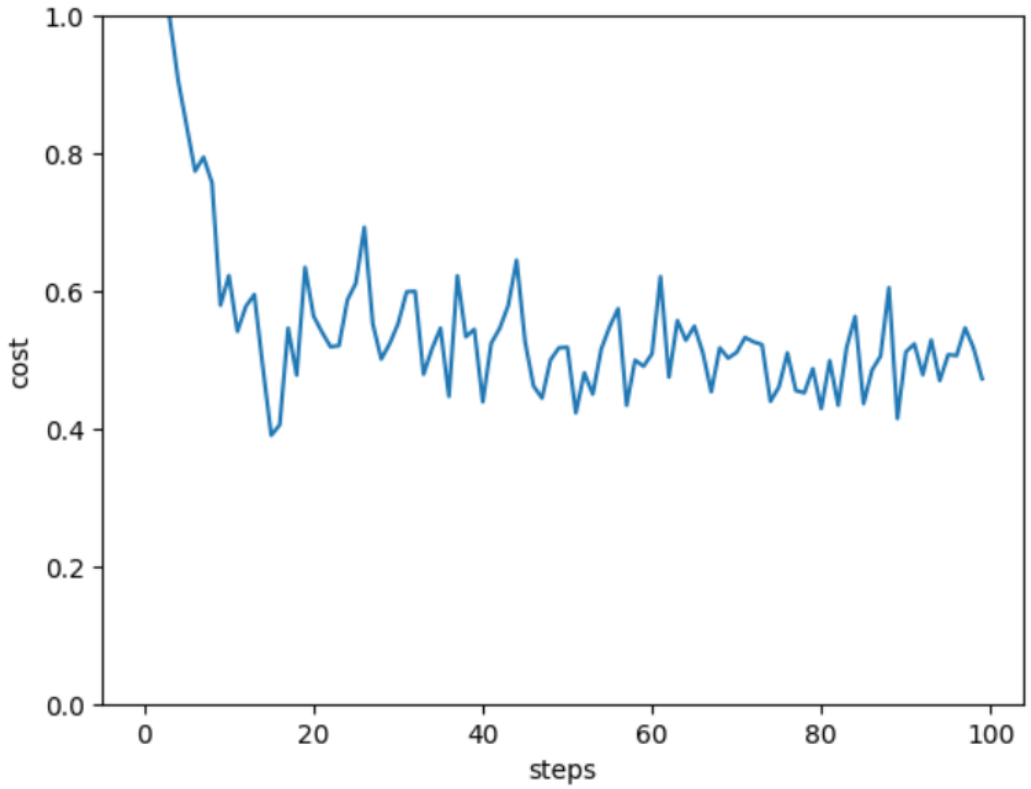
```
''' PREPARACION DE LOS DATOS '''
X, y = load_iris(return_X_y=True)

# pick inputs and labels from the first two classes only,
# corresponding to the first 100 samples
X = X[:100]
y = y[:100]

# scaling the inputs is important since the embedding we use is periodic
scaler = StandardScaler().fit(X)
X_scaled = scaler.transform(X)

# scaling the labels to -1, 1 is important for the SVM and the
# definition of a hinge loss
y_scaled = 2 * (y - 0.5)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled)
```

Los resultados obtenidos son:

- Clasificador variacional

```
n_layers = 2
batch_size = 20
steps = 100
trained_params, trained_bias, loss_history = quantum_model_train(n_layers, steps, batch_size)

pred_test = quantum_model_predict(X_test, trained_params, trained_bias)
print("accuracy on test set:", accuracy_score(pred_test, y_test))

plt.plot(loss_history)
plt.ylim((0, 1))
plt.xlabel("steps")
plt.ylabel("cost")
plt.show()
```

```
step 0 , loss 1.1475412259763673
step 10 , loss 0.6225860003250863
step 20 , loss 0.5636062183855094
step 30 , loss 0.5528368621812871
step 40 , loss 0.4389680760598537
step 50 , loss 0.5181873949784691
step 60 , loss 0.5088495235980648
step 70 , loss 0.5107381548701473
step 80 , loss 0.4288821073281954
step 90 , loss 0.5114232905321632
accuracy on test set: 1.0
```



Como se puede observar el algoritmo realiza un proceso iterativo en el que ajusta el valor de sus parámetros reduciendo el error hasta converger a la solución. A diferencia de otros procesos iterativos, en este casos se observa que existe cierta variabilidad en el valor del error cometido con el paso de las iteraciones. Esto es debido a la naturaleza de la implementación por la que en cada iteración se elige un subconjunto diferente de train sobre el que el algoritmo aprende.

- Clasificador basado en kernels

```
'''ENTRENAMIENTO Y PREDICCION'''
print("Entrenando modelo....")
svm = SVC(kernel=kernel_matrix,C=10.0,gamma=0.0001)
svm.fit(X_train, y_train)
print("Prediciendo valores........")
predictions = svm.predict(X_test)
prec=accuracy_score(predictions, y_test)
print("Precision del modelo ", prec)
```

```
Entrenando modelo....
Prediciendo valores........
Precision del modelo  1.0
```

Como se puede observar el algoritmo utiliza una máquina de vector soporte de la librería scikit-learn junto con el kernel generado por el circuito para entrenar. De nuevo se puede observar que el algoritmo es capaz de aprender a clasificar este tipo de datos en clases.

- Versión cuántica del algoritmo K-vecinos

```
'''ENTRENAMIENTO Y PREDICCION'''
n=2
neigh = KNeighborsClassifier(n_neighbors=n,metric=distance)
print("Modelo KNN con",n,"vecinos")
print("Entrenando modelo......")
neigh.fit(X_train,y_train)
print("Prediciendo valores de test.....")
score=neigh.score(X_test, y_test)
print("Score del modelo: ",score)
```

```
Modelo KNN con 2 vecinos
Entrenando modelo......
Prediciendo valores de test.....
Score del modelo:  1.0
```

Al igual que en el caso anterior, se emplea un algoritmo de scikit learn para entrenar los datos pero la distancia entre elementos para determinar su afinidad a una clase u otra es calculada por el circuito cuántico.
En conclusión y como se puede observar, se ha podido entrenar los 3 modelos anteriores para el conjunto de datos Iris. Estos 3 modelos se emplearán posteriormente para el caso de uso elegido ajustando su estructura y parámetros buscando la configuración óptima de cada uno.
Optimización

Dentro del subproblema de optimización, la solución cuántica elegida es el Solucionador Propio Variacional Cuántico (Variational Quantum Eigensolver, VQE).
VQE es un algoritmo cuántico que utiliza una técnica variacional para encontrar el valor propio mínimo del hamiltoniano de un sistema dado.
Una instancia de VQE requiere la definición de dos subcomponentes algorítmicos: un estado de prueba (también conocido como ansatz) que es un Circuito Cuántico y uno de los optimizadores clásicos. El ansatz se va modificando a través de su conjunto de parámetros por el optimizador, de modo que trabaja hacia un estado, según lo determinado por los parámetros aplicados al ansatz. Esto dará como resultado que se mida el valor esperado mínimo del operador de entrada (Hamiltoniano) .
Se puede proporcionar una matriz opcional de valores de parámetros (a través del initial_point) como punto de partida para la búsqueda del valor propio mínimo. Esta característica es particularmente útil cuando hay razones para creer que el punto de solución está cerca de un punto en particular.
El tamaño del llamado initial_point debe coincidir con el número de parámetros que utilizará el ansatz. En caso de prescindir de estos valores de entrada, VQE tomará como referencia el ansatz para

encontrar el mejor valor basándose a su vez en su el estado inicial dado. Si esto tampoco ofrece ningún resultado, se genera un punto aleatorio dentro de los límites de los parámetros.

```
from qiskit.algorithms.optimizers import OptimizerResult

def my_minimizer(fun, x0, jac=None, bounds=None) -> OptimizerResult:
    # Note that the callable *must* have these argument names!
    # Args:
    #     fun (callable): the function to minimize
    #     x0 (np.ndarray): the initial point for the optimization
    #     jac (callable, optional): the gradient of the objective function
    #     bounds (list, optional): a list of tuples specifying the parameter bounds

    result = OptimizerResult()
    result.x = # optimal parameters
    result.fun = # optimal function value
    return result
```

Respecto a la validación se realiza sobre un experimento inicial de un ejemplo muy sencillo, en el que se utiliza el la librería open-source Qiskit aplicando las funciones cuánticas disponibles para solucionar el problema de optimización en un sencillo grafo no dirigido como el siguiente:

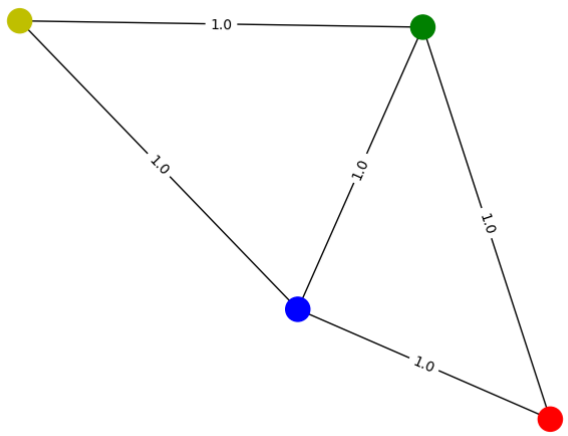


Figura. Grafo de ejemplo para la validación de VQE en el dominio de optimización.

El código para generar dicho grafo es el siguiente:

```python
from __future__ import annotations

import numpy as np
import networkx as nx

num_nodes = 4
w = np.array([[0., 1., 1., 0.],
              [1., 0., 1., 1.],
              [1., 1., 0., 1.],
              [0., 1., 1., 0.]])
G = nx.from_numpy_array(w)
layout = nx.random_layout(G, seed=10)
colors = ['r', 'g', 'b', 'y']
nx.draw(G, layout, node_color=colors)
labels = nx.get_edge_attributes(G, 'weight')
nx.draw_networkx_edge_labels(G, pos=layout, edge_labels=labels);
```

Si intentásemos una aproximación de fuerza bruta, tendríamos que probar de manera exhaustiva todas las parejas de nodos. Para cada pareja, se asigna un 0 si el vértice está en la partición inicial y un 1 si está en la segunda. A continuación se muestra el código que imprime la asignación binaria que satisface la definición de la partición del gráfico y corresponde al número mínimo de aristas cruzadas.

```python
def objective_value(x: np.ndarray, w: np.ndarray) -> float:
    """Compute the value of a cut.
    Args:
        x: Binary string as numpy array.
        w: Adjacency matrix.
    Returns:
        Value of the cut.
    """
    X = np.outer(x, (1 - x))
    w_01 = np.where(w != 0, 1, 0)
    return np.sum(w_01 * X)

def bitfield(n: int, L: int) -> list[int]:
    result = np.binary_repr(n, L)
    return [int(digit) for digit in result]  # [2:] to chop off the "0b" part

# use the brute-force way to generate the oracle
L = num_nodes
max = 2**L
sol = np.inf
for i in range(max):
    cur = bitfield(i, L)

    how_many_nonzero = np.count_nonzero(cur)
    if how_many_nonzero * 2 != L:  # not balanced
        continue

    cur_v = objective_value(np.array(cur), w)
    if cur_v < sol:
        sol = cur_v

print(f'Objective value computed by the brute-force method is {sol}')
```

```
Objective value computed by the brute-force method is 3
```

A continuación llevamos a cabo una prueba del algoritmo VQE mencionado en la que se resuelva el problema mediante computación cuántica:

```
from qiskit.algorithms.minimum_eigensolvers import SamplingVQE
from qiskit.circuit.library import TwoLocal
from qiskit.utils import algorithm_globals

algorithm_globals.random_seed = 10598

optimizer = COBYLA()
ansatz = TwoLocal(qubit_op.num_qubits, "ry", "cz", reps=2, entanglement="linear")
sampling_vqe = SamplingVQE(sampler, ansatz, optimizer)

result = sampling_vqe.compute_minimum_eigenvalue(qubit_op)

x = sample_most_likely(result.eigenstate)

print(x)
print(f"Objective value computed by VQE is {objective_value(x, w)}")
```

```
[0 1 0 1]
Objective value computed by VQE is 3
```

Simulación

Se valida el VQE para encontrar el estado de mínima energía (*ground state*) de la molécula H_2, ya que es muy simple, para demostrar que este algoritmo funciona correctamente y puede ser simulado de principio a fin en una plataforma de simulación cuántica de manera correcta.

Se realiza sobre dos librerías: Qiskit y PennyLane.

- Implementación con Qiskit

Las versiones de las librerías utilizadas en esta implementación son las siguientes: Qiskit, versión 0.39.2; Qiskit Nature, versión 0.4.5; Pyscf, versión 2.1.1.

El primer paso que se debe realizar es el cálculo del hamiltoniano fermiónico correspondiente a la geometría molecular elegida. Esto se puede llevar a cabo con el método ElectronicStructureProblem de Qiskit Nature. En nuestro caso, hemos optado por utilizar un driver molecular basado en Pyscf y con base sto3g. El hamiltoniano obtenido tiene 36 términos, de los cuales mostramos una versión truncada a continuación, tal como se obtiene en Qiskit:

-1.2533097866459775 * (+_0 -_0)
+ -0.47506884877217725 * (+_1 -_1)
+ -1.2533097866459775 * (+_2 -_2)
+ -0.47506884877217725 * (+_3 -_3)
+ -0.33737796340722415 * (+_0 +_0 -_0 -_0)
+ -0. ...

Este hamiltoniano no puede utilizarse directamente con el método VQE, que espera un hamiltoniano expresado como combinación lineal de productos tensoriales de matrices X, Y y Z de Pauli. Para conseguirlo, hemos usado la transformación de Jordan-Wigner. Como resultado, hemos obtenido el siguiente hamiltoniano:

-0.8121706072487122 * IIII
+ 0.17141282644776895 * IIIZ
- 0.2234315369081349 * IIZI
+ 0.17141282644776906 * IZII
- 0.2234315369081349 * ZIII
+ 0.12062523483390418 * IIZZ
+ 0.16868898170361207 * IZIZ
+ 0.04530261550379923 * YYYY
+ 0.04530261550379923 * XXYY
+ 0.04530261550379923 * YYXX
+ 0.04530261550379923 * XXXX
+ 0.1659278503377034 * ZIIZ
+ 0.1659278503377034 * IZZI
+ 0.1744128761226159 * ZIZI
+ 0.12062523483390418 * ZZII

- Implementación con PennyLane

Para la implementación del algoritmo VQE en PennyLane, se ha utilizado la versión 0.26 de la librería. Se ha fijado la misma geometría de molécula que en el caso de Qiskit, con la salvedad de que PennyLane espera que los valores de distancia sean expresados en unidades atómicas, por lo que se ha realizado la conversión de ángstroms a esta unidad, obteniendo un valor de 1.3228, aproximadamente.
El método qchem.molecular_hamiltonian permite obtener directamente el hamiltoniano expresado como combinación lineal de producto tensorial de puertas de Pauli. El resultado obtenido ha sido el siguiente:

(-0.22343155727095726) [Z2]
+ (-0.22343155727095726) [Z3]
+ (-0.09706620778626623) [I0]
+ (0.17141283498167342) [Z1]
+ (0.1714128349816736) [Z0]
+ (0.12062523781179485) [Z0 Z2]
+ (0.12062523781179485) [Z1 Z3]
+ (0.16592785242008765) [Z0 Z3]
+ (0.16592785242008765) [Z1 Z2]
+ (0.16868898461469894) [Z0 Z1]
+ (0.17441287780052514) [Z2 Z3]
+ (-0.04530261460829278) [Y0 Y1 X2 X3]
+ (-0.04530261460829278) [X0 X1 Y2 Y3]
+ (0.04530261460829278) [Y0 X1 X2 Y3]

+ (0.04530261460829278) [X0 Y1 Y2 X3]

Este hamiltoniano es diferente del obtenido en el caso de Qiskit porque aquí se tiene en cuenta la energía total de la molécula, no solo la parte electrónica.
PennyLane no proporciona una implementación de la forma variacional EfficientSU2, por lo que se ha optado por realizar una implementación propia. En cualquier caso, el circuito considerado es el mismo que el mostrado en el caso de Qiskit. Para la minimización de los parámetros, se ha usado el método minimize proporcionado por la librería scipy.optimize, que por defecto usa el algoritmo BFGS. La inicialización de los parámetros se ha llevado a cabo como en la implementación en Qiskit: con el método np.random.random y semilla 1234.
Los resultados obtenidos para los valores óptimos de los parámetros son los siguientes:

[2.25573350e-01 3.14158133e+00 2.10368123e-07 -2.06643452e-06
 -2.71658811e-03 -7.94108629e-01 4.52509473e-01 6.17684341e-01
  3.14158763e+00 6.28319394e+00 3.14158400e+00 3.14160975e+00
  2.21494216e-01 5.01301650e-01 6.51838422e-01 7.36619383e-02]

En cuanto a la energía obtenida, ha sido de -1.1372838, que coincide con la energía total obtenida con la implementación realizada en Qiskit.

PT4: *Framework* de simulación cuántica

Este paquete de trabajo se organiza en torno a tres tareas, de las cuales una (T4.1) ya había sido finalizada en su totalidad en la anualidad 2021, una segunda (T4.2) se inició en 2021 y terminó en esta anualidad 2022. La restante (T4.3), se iniciará y finalizará a lo largo de la última anualidad.

- T4.1: Estudio de simuladores cuánticos (terminada en 2021).
- T4.2: Integración de simuladores cuánticos en el hardware (iniciada en 2021 y terminada en 2022).
- T4.3: Monitorización de *framework* (a comenzar en 2023).

A continuación, se describen las acciones desarrolladas en la tarea T4.2 que han dado conclusión a la misma cuyas actividades se iniciaron en 2021.

T4.2: Integración de simuladores cuánticos en el hardware

La tarea 4.2 comprende los trabajos de integración de un subconjunto de los *frameworks* software identificados en T4.1 en el contexto de una infraestructura hardware de altas prestaciones. De esta manera, se ha habilitado la simulación de circuitos cuánticos con un alto número de qubits.

Los trabajos de esta tarea se han concentrado en dos focos:

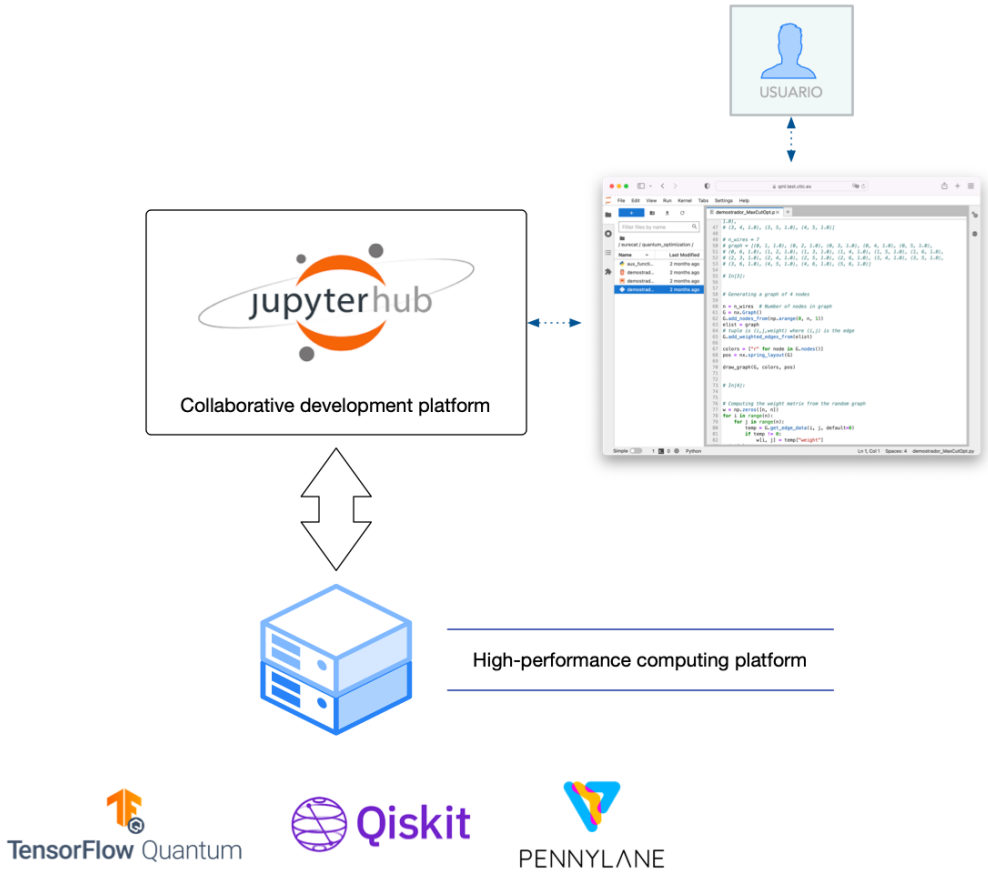
- Desarrollo de pruebas de concepto comparativas para obtención de impresiones de los *frameworks*. Estas son de gran valor, ya que suelen existir detalles importantes que son difíciles de detectar solamente a través de la lectura de documentación. Además, se pueden obtener datos de rendimiento que informen la inversión de un mayor esfuerzo en un *framework* en concreto.

- Diseño e implementación de un componente software reusable y colaborativo para desarrollo de programas de simulación cuántica. Este software es el resultado principal de la tarea.

El componente en cuestión es una solución para el desarrollo y la documentación colaborativa en el ecosistema de Python. Se basa en dos herramientas clave: Jupyter y JupyterHub. Jupyter es conocido por ser una herramienta altamente flexible y versátil que permite a los usuarios crear documentos interactivos que combinan bloques de texto con código fuente ejecutable. Es ampliamente utilizado en la comunidad científica y educativa.

Por otro lado, JupyterHub es una capa adicional que permite la gestión de múltiples usuarios en un único servidor. Con JupyterHub, los usuarios pueden tener acceso a recursos compartidos en entornos aislados, lo que garantiza la seguridad y la privacidad de los datos. Esta herramienta utiliza una extensión de Docker que permite una gestión más sencilla, eficiente y escalable de los entornos aislados. De esta manera, se facilita la gestión, la repetibilidad y la reutilización de los recursos compartidos.

La figura a continuación muestra un diagrama de esta arquitectura basada en Jupyter y JupyterHub.



Los entornos de usuario vienen preconfigurados e integrados con los frameworks de simulación cuántica identificados en la anualidad pasada 2021 (véase el documento E4.1). Estos frameworks se describen a continuación:

- **PennyLane**. Desarrollado por Xanadu, es un framework de código abierto para el aprendizaje automático cuántico y la computación cuántico-clásica. Proporciona una interfaz unificada con las plataformas de hardware cuántico y simuladores más populares, así como un conjunto de herramientas para la diferenciación y optimización de circuitos cuánticos. Con PennyLane, los usuarios pueden escribir algoritmos cuánticos en Python y ejecutarlos en hardware cuántico real o simulaciones, independientemente de la plataforma de hardware subyacente. PennyLane se basa en un constructo denominado la función cuántica, donde los algoritmos cuánticos se representan como funciones diferenciables. Esto permite el uso de métodos de optimización basados en gradientes para entrenar circuitos cuánticos.
- **Qiskit**. Desarrollado por IBM. Proporciona un conjunto completo de herramientas y recursos para crear, ejecutar y visualizar circuitos y algoritmos cuánticos. Qiskit incluye una serie de abstracciones de alto nivel, tales como los registros y los circuitos cuánticos, que facilitan el trabajo con algoritmos cuánticos, proporcionando una representación natural e intuitiva de los estados y operaciones. Además, Qiskit proporciona una serie de algoritmos y

herramientas de dominio específico para la corrección de errores cuánticos, la optimización cuántica y el aprendizaje automático cuántico.

- **Cirq**. Desarrollado por Google. Presenta una API similar a la de Qiskit, basada en los conceptos de Circuits (circuitos) y Gates (puertas) cuánticas. Un Circuit representa una configuración específica de qubits y puertas, mientras que las Gates se pueden aplicar a los qubits para generar operaciones. Un conjunto de operaciones que tienen lugar al mismo tiempo se conoce como un Moment. Además, existen módulos externos compatibles con Cirq que ofrecen algoritmos específicos para diferentes aplicaciones, al igual que en Qiskit.
- **TensorFlow Quantum**. TFQ es una capa construida encima de TensorFlow que reutiliza las primitivas de Keras para proporcionar una API familiar a científicos e investigadores que están acostumbrados a trabajar en el dominio de la computación cuántica. En términos de funcionalidad puede ser comparable al resto de opciones, donde TFQ destaca significativamente es precisamente en esta familiaridad y la potencia que brinda el ecosistema TensorFlow.

En materia de despliegue y administración, se ha adoptado una aproximación basada en las herramientas Makefile, Docker y Docker Compose. Makefile es una de las herramientas más comunes para la automatización y gestión de tareas en Linux. Docker es el producto de tecnologías de contenedores más popular, que proporciona un conjunto de herramientas bien integradas para la gestión del ciclo de vida de las imágenes y los contenedores. Por otra parte, Docker Compose es una de estas herramientas del ecosistema Docker para la gestión de cargas de trabajo multi-contenedor.

De manera más específica, cuando se ejecuta el comando de inicio del despliegue se llevan a cabo los siguientes pasos:

1. Se genera una clave segura para configurar la base de datos PostgreSQL que contiene las entidades de JupyterHub, como por ejemplo los usuarios y las contraseñas.
2. Se crea la red Docker sobre la que se van a desplegar los entornos de desarrollo de los usuarios. Es decir, los contenedores aislados que contienen la pila completa de frameworks de simulación cuántica.
3. Se construyen las imágenes que definen los entornos de desarrollo de los usuarios. Esta imagen es uno de los componentes más críticos, ya que define el entorno software al que va a tener acceso el usuario para desarrollo y pruebas.
4. Finalmente, se despliega el stack que define las imágenes de JupyterHub y la base de datos. Este stack contiene básicamente los servicios software junto con su configuración de puertos, volúmenes y entorno.

PT5: Demostración de casos de uso

Este paquete de trabajo se organiza en torno a tres tareas, una de las cuales (T5.1) se inició y concluyó en esta anualidad 2022, la T5.2 se inició en 2022 y terminará en la anualidad 2023 y la (T5.3), se iniciará y finalizará a lo largo de la última anualidad.

- T5.1: Definición de casos de uso (iniciada y terminada en 2022).
- T5.2: Obtención y preparación de los datos de partida (iniciada en 2022 y por finalizar en 2023).
- T5.3: Pruebas de laboratorio (a comenzar en 2023).

A continuación, se describen brevemente las acciones desarrolladas en la tarea T5.1, recogidas en su totalidad en el E5.1 y se muestra el avance conseguido en la T5.2 que terminará en la anualidad 2023.

T5.1: Definición de casos de uso

En la memoria de la propuesta de proyecto ALCATRAZ, ya se detallaban unos posibles casos de uso que serían de especial interés para la industria. Aunque se dejaba la puerta abierta a poder explorar otras opciones durante el transcurso del proyecto, los algoritmos clásicos han sido ejecutados sobre esos ejemplos debido a su mayor madurez tecnológica respecto a la computación cuántica.

En esta tarea, se terminan de definir los casos de uso específicos que se van a plantear en ALCATRAZ asociados a los retos a los que se enfrenta la industria asturiana que a su vez están enmarcados como subproblemas matemáticos. Están recogidos en la siguiente tabla:

Subproblemas matemáticos	Retos de la industria	Casos de uso
Clasificación	Detección de anomalías	Anticipación a ciberataques
Optimización	Optimización de procesos	Optimización de rutas de transporte
Simulación	Diseño de nuevos productos	Optimización geométrica de moléculas

Para cada caso de uso, se describe el planteamiento matemático del caso de uso específico. En el caso de uso de **clasificación**, anticipación a ciberataques, el conjunto de datos a utilizar corresponde con una serie de interacciones con un servidor, las cuales se clasifican en ciberataques DDoS (del inglés, *Distributed Denial-of-Service*) o benignas similar. Debido a las limitaciones técnicas y de tiempo que supondría utilizar el conjunto completo de los datos para construir el modelo cuántico, se ha decidido utilizar solo uno de los subconjuntos de datos que conforman el conjunto total de datos de ciberataques, el conjunto de datos NetBios.
Se plantea el uso de los tres algoritmos cuánticos validados: el clasificador variacional, el clasificador basado en kernels y la versión cuántica del algoritmo k-vecinos. Para cada uno de ellos se buscará la configuración óptima para el problema en cuestión de cara a realizar una comparativa de rendimiento y determinar cuál de los 3 es el más eficiente.
Los principales puntos que se van a tratar son:

- Determinar el optimizador y ratio de aprendizaje para el clasificador variacional.
- Determinar el número óptimo de unidades de procesamiento para mejorar el rendimiento de predicción del algoritmo K-vecinos.
- Mejorar el rendimiento del algoritmo basado en Kernels teniendo en cuenta el proceso de creación de la matriz o kernel.

En el caso de uso de **optimización** de rutas de transporte, los experimentos se realizan en dos pasos: generación de distancias aleatorias y datos de producción reales. Primero, se genera una matriz de distancias aleatorias y, después, se procede al uso de datos reales para la búsqueda de las rutas óptimas.

En el caso de **simulación**, para llevar a cabo la optimización geométrica de moléculas que es el caso de uso elegido, se parte de una geometría fija, se designan las distancias entre los átomos que forman la molécula y se busca el estado de mínima energía, también llamado estado fundamental (*ground state*), ya que la geometría con una energía menor será la más disposición espacial estable y, por tanto, la que va a adoptar la molécula, resolviendo así el problema.
El proceso para optimizar la geometría de la molécula es el siguiente:

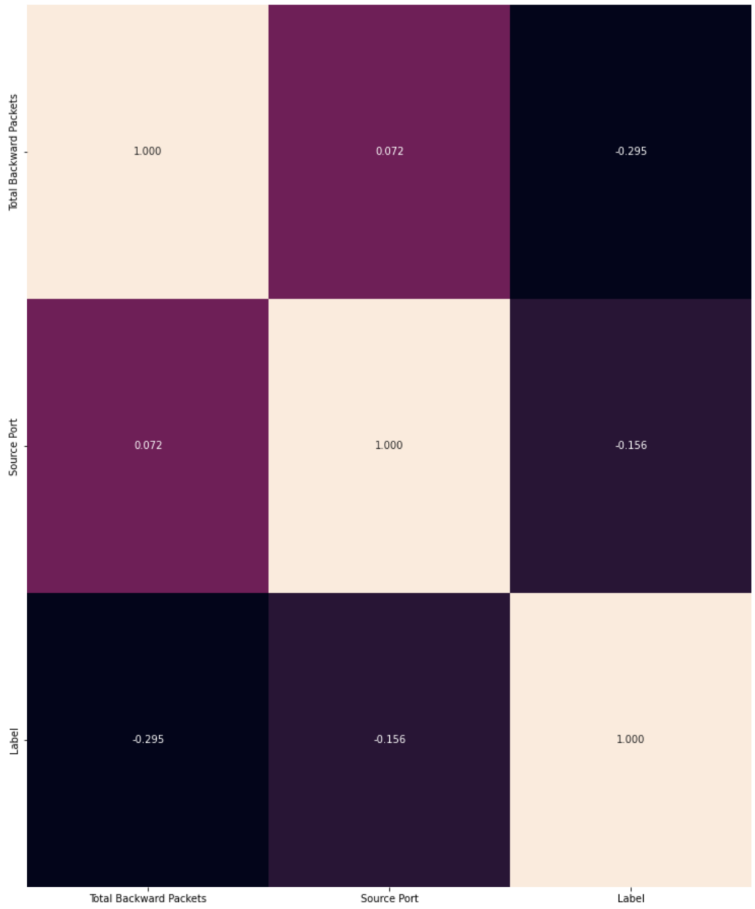
1. Calcular la energía de la geometría actual utilizando la función de onda.
2. Modificar la geometría de la molécula.
3. Calcular la nueva energía de la molécula modificada utilizando la función de onda.
4. Evaluar el gradiente energético para determinar si la nueva geometría es una optimización (es decir, si se ha alcanzado el mínimo energético).
5. Si la energía ha disminuido, repetir desde el paso 2, de lo contrario, deshacer la modificación y modificar la geometría en otra dirección.
6. Repetir hasta alcanzar el mínimo energético.

Se plantea llevar a cabo este experimento para las moléculas de agua y metano.

T5.2: Obtención y preparación de los datos de partida

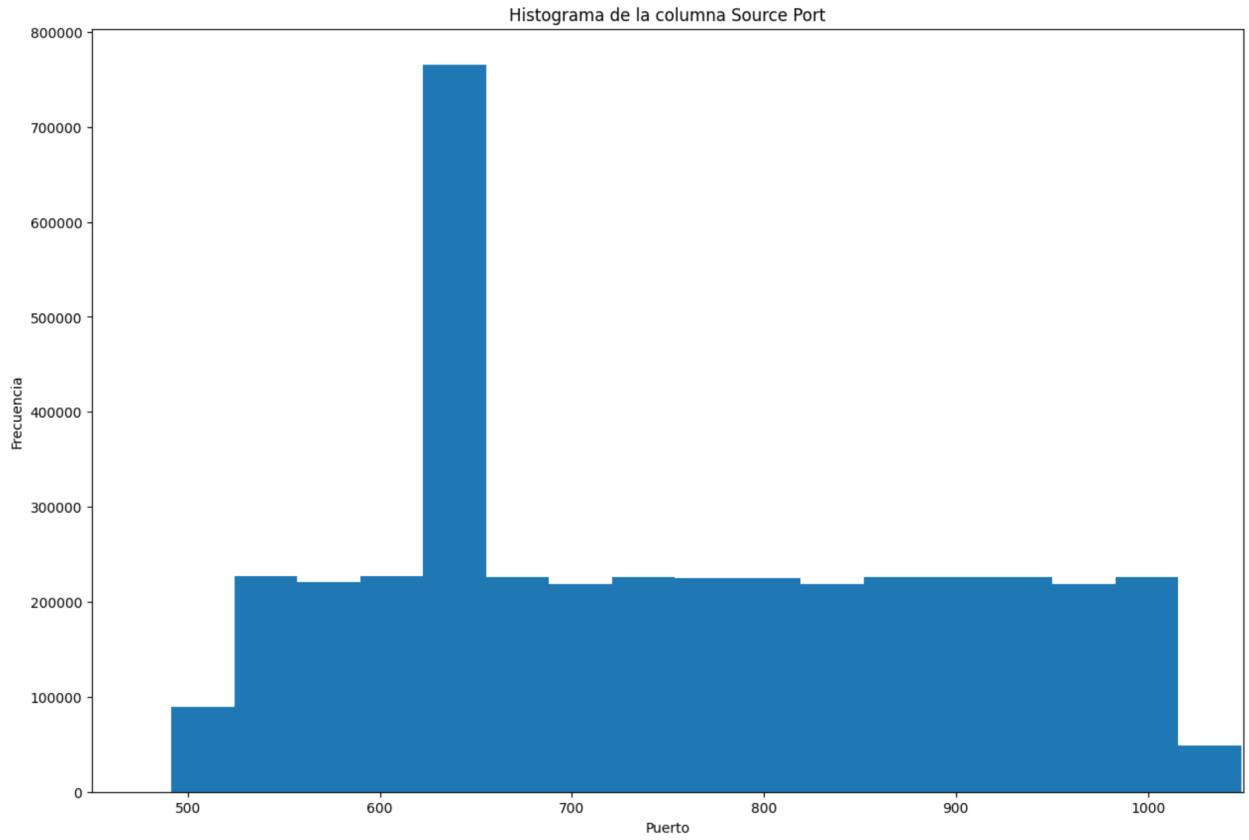
A partir del conjunto de datos de detección de ciberataques se realizó un preprocesamiento de los datos para ajustarse a los modelos cuánticos. Sin embargo, estos datos contaban con cierto sesgo y por ello se han seleccionado otro conjunto de variables de entrenamiento teniendo en cuenta la matriz de correlaciones.

De entre el conjunto de posibles variables se han seleccionado puerto de origen (Source Port) y el número de paquetes recibidos (Total Backward Packets). Como apoyo visual se muestra la matriz de correlaciones con las nuevas variables:



Como se puede observar y comparándola con la matriz de correlaciones oficial, se ha reducido considerablemente el sesgo anterior y no existe una relación tan fuerte entre las variables de entrenamiento y la variable objetivo.

Si se observa la distribución de la nueva variable introducida, Source Port, se puede ver que su distribución de valores es más uniforme:



De esta manera el conjunto de datos queda conformado por 2 variables de entrenamiento con valores numéricos enteros y una tercera variables binaria que será la variable objetivo a predecir por los modelos de clasificación clásicos y cuánticos y se podrá comparar su rendimiento tanto por la calidad de sus predicciones como el tiempo requerido para entrenar y predecir.

Justificación 2023

Tal y como se puede apreciar en el cronograma mostrado, a lo largo de la anualidad 2023 se han terminado los paquetes de trabajo restantes: el PT4 (*Framework* de simulación cuántica), el PT5 (Demostración de casos de uso) y PT6 (Validación de la viabilidad de las soluciones cuánticas).

A continuación, se describen las tareas realizadas en este último hito de trabajo y los resultados conseguidos.

PT4: *Framework* de simulación cuántica

Este paquete de trabajo se organiza en torno a tres tareas, de las cuales una (T4.1) ya había sido finalizada en su totalidad en la anualidad 2021, una segunda (T4.2) se inició en 2021 y terminó en 2022 y la restante (T4.3) que se inició y finalizó a lo largo de la última anualidad del proyecto, 2023.

- T4.1: Estudio de simuladores cuánticos (terminada en 2021).

- T4.2: Integración de simuladores cuánticos en el hardware (terminada en 2022).
- T4.3: Monitorización del *framework* (iniciada y terminada en 2023).

A continuación, se describen las actividades desarrolladas en la tarea T4.3.

T4.3: Monitorización del *framework*

Las actividades llevadas a cabo en esta tarea son las siguientes:

Taskfile en lugar de Makefile

Makefile y Taskfile sirven para automatizar tareas en el desarrollo de software. Makefile es una herramienta tradicional y sobradamente probada, pero Taskfile ofrece una sintaxis más clara y funcionalidades adicionales que simplifican significativamente tareas como la definición de comandos que dependen del sistema operativo, o la definición de precondiciones y requisitos de ejecución del árbol de tareas. Es por esto que, para facilitar la mantenibilidad futura del entorno de simulación de ALCATRAZ, se decidió migrar de Makefile a Taskfile para la gestión de tareas de despliegue y mantenimiento.
A continuación se incluye la definición en formato YAML de las tareas de ALCATRAZ. Esto sirve como contraste al fichero Makefile.

```
version: "3"

dotenv: [.env.local, .env]

tasks:
  debug-env:
    silent: true
    cmds:
      - echo "IMAGE_RUNTIME={{.IMAGE_RUNTIME}}"
      - echo "JUPYTERHUB_VERSION={{.JUPYTERHUB_VERSION}}"
      - echo "NVIDIA_IMAGE_TAG={{.NVIDIA_IMAGE_TAG}}"

  check-runtime:
    cmds:
      - "{{.ROOT_DIR}}/check-runtime.sh"

  create-network:
    cmds:
      - docker network create {{.DOCKER_NETWORK_NAME}}
    status:
      - docker network inspect {{.DOCKER_NETWORK_NAME}}

  create-postgres-secret:
    cmds:
      - >
        echo "POSTGRES_PASSWORD=$(openssl rand -hex 32)" >
        {{.ROOT_DIR}}/secrets/postgres.env
    status:
```

```
    - test -f {{.ROOT_DIR}}/secrets/postgres.env

build-notebook-image:
  cmds:
    - >
      docker buildx build --platform linux/amd64
      -t {{.LOCAL_NOTEBOOK_IMAGE}}
      --build-arg JUPYTERHUB_VERSION={{.JUPYTERHUB_VERSION}}
      --build-arg NVIDIA_IMAGE_TAG={{.NVIDIA_IMAGE_TAG}}
      --build-arg NOTEBOOK_USER={{.NOTEBOOK_USER}}
      {{.ROOT_DIR}}/singleuser

build:
  desc: Build the notebook image and the Compose stack
  deps:
    - create-postgres-secret
    - create-network
    - build-notebook-image
  cmds:
    - docker compose build

up:
  desc: Start the Compose stack
  deps:
    - build
  cmds:
    - docker compose up --build -d

down:
  desc: Stop the Compose stack
  cmds:
    - docker compose down

clean:
  desc: Stop the Compose stack and remove all traces
  cmds:
    - docker compose down -v
    - cmd: docker network rm {{.DOCKER_NETWORK_NAME}}
      ignore_error: true
    - rm -f {{.ROOT_DIR}}/secrets/postgres.env
```

Integración de CUDA

CUDA es un framework para computación paralela creado por NVIDIA, diseñado para utilizar las capacidades singulares de las tarjetas gráficas (GPUs). Esta tecnología habilita mejoras significativas en el rendimiento de los cálculos al utilizar las GPUs para tareas de computación generales. En particular, CUDA es crucial para facilitar simulaciones avanzadas de computación cuántica, ya que

permite implementar de manera eficiente algoritmos cuánticos complejos y simular sistemas cuánticos a gran escala aprovechando las capacidades de paralelización masiva que ofrecen las GPUs. Las herramientas de despliegue del entorno de simulación cuántica de ALCATRAZ se actualizaron para aprovechar las posibilidades de CUDA, si es que están disponibles en el servidor sobre el que se despliega.
Para conseguir esto, en primer lugar, se incluyó un script que comprueba automáticamente la disponibilidad del runtime de Docker de NVIDIA. En caso de que no esté disponible, se avisa al administrador. Esto es importante porque la configuración del runtime es algo que puede ser fácilmente pasado por alto. Al utilizar el runtime por defecto (runc), los procesos funcionan igualmente sin problemas. Sin embargo, la diferencia es que la GPU no está disponible y las simulaciones se ven forzadas a ejecutarse en la CPU de manera silenciosa.

```
#!/usr/bin/env bash

set -e
set -x

RUNTIMES=$(docker info -f "{{.Runtimes}}")
NVIDIA="nvidia"

GREEN='\033[0;32m'
YELLOW='\033[0;33m'
NC='\033[0m'

if [[ ${RUNTIMES} =~ .*"${NVIDIA}".* ]]; then
        # trunk-ignore(shellcheck/SC2059)
        printf "${GREEN}The NVIDIA Container Runtime (\"${NVIDIA}\") is available.${NC}\n"
else
        # trunk-ignore(shellcheck/SC2059)
        printf "${YELLOW}The NVIDIA Container Runtime (\"${NVIDIA}\") does not seem to be available in this system. Please make sure to update the configuration to use another runtime.${NC}\n"
fi
```

En segundo lugar, si el runtime Docker de NVIDIA está disponible y apropiadamente configurado, se solicitan las capacidades de la GPU en la configuración del gestor de contenedores Docker de usuario. Este gestor (spawner) es el componente que se encarga de crear y destruir los contenedores individuales de usuario en los que se ejecutan los experimentos.

```
[...]

the_runtime = os.environ.get("IMAGE_RUNTIME", "nvidia")
count_gpus = int(os.environ.get("DEVICE_REQUESTS_GPUS", 1))
device_requests = []

if the_runtime == "nvidia":
```

```
    device_requests.append(
        docker.types.DeviceRequest(count=count_gpus, capabilities=[["gpu"]]))

c.DockerSpawner.extra_host_config = {
    "runtime": the_runtime,
    "mem_limit": "512g",
    "mem_swappiness": 0,
    "device_requests": device_requests,
}
```

Finalmente también se ha actualizado la imagen base de los contenedores de experimentación de usuario. Los detalles de esto se incluyen en la sección a continuación.

Imagen de contenedor de usuario

Los cambios para la integración de CUDA requieren actualizar la imagen del contenedor de experimentación de usuario. Aunque el sistema operativo esté correctamente configurado con CUDA, el contenedor no podrá usar la GPU si no se utiliza una imagen derivada de nvidia/cuda (o alguna imagen configurada de manera similar, siendo nvidia/cuda la opción más sencilla).

En el archivo Dockerfile que se muestra a continuación, se observa el uso de la imagen base nvidia/cuda. Además, se instalan las dependencias necesarias, como las bibliotecas libcudnn, que dependen de la versión específica de CUDA utilizada. Es interesante destacar que la configuración de CUDA es un proceso delicado y que las versiones deben ser muy específicas, siguiendo estrictamente las indicaciones de la documentación de NVIDIA.

Esta nueva imagen incluye una mejora adicional: se evita el uso del usuario root, creando un usuario con los permisos adecuados y mínimos para ejecutar los experimentos.

```
ARG NVIDIA_IMAGE_TAG
FROM nvidia/cuda:$NVIDIA_IMAGE_TAG

ARG JUPYTERHUB_VERSION
ARG NOTEBOOK_USER=jovyan

RUN apt-get update -y && DEBIAN_FRONTEND=noninteractive apt-get install --no-install-recommends -y \
    software-properties-common \
    git \
    curl \
    build-essential \
    pkg-config \
    libcairo2-dev \
    libdbus-glib-1-dev \
    libgirepository1.0-dev

RUN add-apt-repository -y ppa:deadsnakes/ppa

RUN apt-get update -y && DEBIAN_FRONTEND=noninteractive apt-get install -y --no-install-recommends \
    python3.8-dev \
```

```
    python3.8-venv \
    python3.8-distutils

RUN curl --silent -o get-pip.py https://bootstrap.pypa.io/get-pip.py && \
    python3.8 get-pip.py && \
    rm get-pip.py

RUN python3.8 -m pip install --no-cache \
    jupyterhub==$JUPYTERHUB_VERSION \
    jupyterlab==3.*

RUN CUDA_VERSION=$(python3.8 -c "import os;
print('.'.join(os.environ['CUDA_VERSION'].split('.')[:2]))") && \
    apt-get update -y && apt-get install --no-install-recommends -y \
    libcudnn8=*+cuda${CUDA_VERSION} \
    libcudnn8-dev=*+cuda${CUDA_VERSION}

COPY requirements.txt /tmp/
RUN pip3.8 install --no-cache-dir --no-deps -U -r /tmp/requirements.txt

COPY jupyter_server_config.py /etc/jupyter/jupyter_server_config.py

ENV NB_UID=1000
ENV NB_GID=100
ENV HOME=/home/$NOTEBOOK_USER

RUN useradd -r -u $NB_UID -g $NB_GID $NOTEBOOK_USER && \
    mkdir -p $HOME && \
    chown -R $NB_UID:$NB_GID $HOME

USER $NOTEBOOK_USER
WORKDIR $HOME

RUN mkdir .local
COPY --chown=$NB_UID:$NB_GID examples /home/$NOTEBOOK_USER/work
RUN mkdir -p /home/$NOTEBOOK_USER/work/user

HEALTHCHECK NONE

CMD ["jupyterhub-singleuser"]
```

Ajuste de las versiones de paquetes

El dominio de la computación cuántica todavía se encuentra en un estado bastante inestable, lo que se refleja a su vez en la estabilidad de las librerías de software utilizadas para programar circuitos y simulaciones. Con el paso del tiempo, han surgido actualizaciones nuevas que resultan incompatibles con la configuración inicial de ALCATRAZ, causando fallos en el proceso de despliegue debido a estas incompatibilidades.

Para solucionar esto, se revisaron detalladamente todas las dependencias y se construyó un archivo de requisitos explícito y completo. Este archivo de requisitos servirá para garantizar despliegues reproducibles en el futuro, evitando los problemas derivados de la disponibilidad de nuevas versiones.

PT5: Demostración de casos de uso

Este paquete de trabajo se organiza en torno a tres tareas, una de las cuales (T5.1) se inició y concluyó en la anualidad 2022, la T5.2 se inició en 2022 y terminó en la anualidad 2023 y la (T5.3), que se inició y finalizó a lo largo de la última anualidad.

- T5.1: Definición de casos de uso (iniciada y terminada en 2022).
- T5.2: Obtención y preparación de los datos de partida (iniciada en 2022 y terminada en 2023).
- T5.3: Pruebas de laboratorio (iniciada y terminada en 2023).

A continuación, se describen las acciones desarrolladas en las tareas T5.2 y T5.3, ambas terminadas en la anualidad 2023.

T5.2: Obtención y preparación de los datos de partida

Los datasets seleccionados y preparados para la aplicación de los algoritmos desarrollados se muestran a continuación:

Clasificación

Para el caso de clasificación, se ha utilizado el conjunto de datos DrDoS_NetBIOS que contiene una serie de trazas de red etiquetadas en **ciberataques** del tipo Dos o en benignas. Para utilizar este conjunto de datos empezamos importando las librerías necesarias y leyendo el fichero csv.

```
import pandas as pd
import numpy as np
```

```
df=pd.read_csv('DrDoS_NetBIOS.csv').drop('Unnamed: 0',axis=1)
df
```

```
/tmp/ipykernel_10946/2898222060.py:1: DtypeWarning: Columns (85) have mixed types. Specify dtype option on import or set low_memory=False.
  df=pd.read_csv('DrDoS_NetBIOS.csv').drop('Unnamed: 0',axis=1)
```

	Flow ID	Source IP	Source Port	Destination IP	Destination Port	Protocol	Timestamp	Flow Duration	Total Fwd Packets	Total Backward Packets	...	Active Std	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	Idle Min	SimillarHTTP	Inbound	Label
0	172.16.0.5-192.168.50.1-34012-2334-17	172.16.0.5	34012	192.168.50.1	2334	17	2018-12-01 11:47:08.463789	1	2	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	1	DrDoS_NetBIOS
1	172.16.0.5-192.168.50.1-34013-50170-17	172.16.0.5	34013	192.168.50.1	50170	17	2018-12-01 11:47:08.464316	1	2	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	1	DrDoS_NetBIOS
2	172.16.0.5-192.168.50.1-34014-61534-17	172.16.0.5	34014	192.168.50.1	61534	17	2018-12-01 11:47:08.464472	1	2	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	1	DrDoS_NetBIOS
3	172.16.0.5-192.168.50.1-34015-8930-17	172.16.0.5	34015	192.168.50.1	8930	17	2018-12-01 11:47:08.464520	1	2	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	1	DrDoS_NetBIOS
4	172.16.0.5-192.168.50.1-34016-33040-17	172.16.0.5	34016	192.168.50.1	33040	17	2018-12-01 11:47:08.464925	2	2	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	1	DrDoS_NetBIOS
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4094981	172.16.0.5-192.168.50.1-525-3770-17	172.16.0.5	525	192.168.50.1	3770	17	2018-12-01 12:00:13.902190	2	2	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	1	DrDoS_NetBIOS
4094982	172.16.0.5-192.168.50.1-648-45456-17	172.16.0.5	648	192.168.50.1	45456	17	2018-12-01 12:00:13.902193	47	2	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	1	DrDoS_NetBIOS
4094983	172.16.0.5-192.168.50.1-526-53287-17	172.16.0.5	526	192.168.50.1	53287	17	2018-12-01 12:00:13.902542	1	2	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	1	DrDoS_NetBIOS
4094984	172.16.0.5-192.168.50.1-527-34907-17	172.16.0.5	527	192.168.50.1	34907	17	2018-12-01 12:00:13.902674	1	2	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	1	DrDoS_NetBIOS
4094985	172.16.0.5-192.168.50.1-648-54457-17	172.16.0.5	648	192.168.50.1	54457	17	2018-12-01 12:00:13.902732	1	2	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	1	DrDoS_NetBIOS

4094986 rows × 87 columns

El siguiente paso consistió en limpiar el dataset. Este proceso incluye:

- Eliminación de espacios en los nombres de las columnas de manera que haya menos problemas al acceder a determinadas variables.
- Sustitución de posibles valores infinitos o desconocidos por valores Nan de Numpy
- Eliminación de todas las instancias con valores Nan.

```
df.columns = df.columns.str.strip()
df=df.replace([np.inf, -np.inf], np.nan)
df=df.dropna()
df=df.loc[:, (df != 0).any(axis=0)]
```

El fichero original posee 87 columnas incluyendo la de la variable objetivo “Label”. Sin embargo, la mayoría de estas columnas poseen un único valor repetido o no aporta información relevante para el problema por lo que se seleccionó un subconjunto de estas variables reduciendo de esta forma la dimensionalidad del dataset.

```
use_columns = ["Flow ID", "Source IP", "Source Port", "Destination IP", "Destination Port", "Timestamp", "Flow Duration", "Total Fwd Packets", "Total Backward Packets", "Fwd IAT Mean", "Fwd IAT Std",
               "Fwd IAT Min", "Fwd Packets/s", "Bwd Packets/s","SYN Flag Count", "RST Flag Count", "ACK Flag Count", "URG Flag Count", "Label"]
df=df[use_columns]
df
```

	Flow ID	Source IP	Source Port	Destination IP	Destination Port	Timestamp	Flow Duration	Total Fwd Packets	Total Backward Packets	Fwd IAT Mean	Fwd IAT Std	Fwd IAT Min	Fwd Packets/s	Bwd Packets/s	SYN Flag Count	RST Flag Count	ACK Flag Count	URG Flag Count	Label
0	172.16.0.5-192.168.50.1-34012-2334-17	172.16.0.5	34012	192.168.50.1	2334	2018-12-01 11:47:08.463789	1	2	0	1.0	0.0	1.0	2.000000e+06	0.0	0	0	0	0	DrDoS_NetBIOS
1	172.16.0.5-192.168.50.1-34013-50170-17	172.16.0.5	34013	192.168.50.1	50170	2018-12-01 11:47:08.464316	1	2	0	1.0	0.0	1.0	2.000000e+06	0.0	0	0	0	0	DrDoS_NetBIOS
2	172.16.0.5-192.168.50.1-34014-61534-17	172.16.0.5	34014	192.168.50.1	61534	2018-12-01 11:47:08.464472	1	2	0	1.0	0.0	1.0	2.000000e+06	0.0	0	0	0	0	DrDoS_NetBIOS
3	172.16.0.5-192.168.50.1-34015-8930-17	172.16.0.5	34015	192.168.50.1	8930	2018-12-01 11:47:08.464520	1	2	0	1.0	0.0	1.0	2.000000e+06	0.0	0	0	0	0	DrDoS_NetBIOS
4	172.16.0.5-192.168.50.1-34016-33040-17	172.16.0.5	34016	192.168.50.1	33040	2018-12-01 11:47:08.464925	2	2	0	2.0	0.0	2.0	1.000000e+06	0.0	0	0	0	0	DrDoS_NetBIOS
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4094981	172.16.0.5-192.168.50.1-525-3770-17	172.16.0.5	525	192.168.50.1	3770	2018-12-01 12:00:13.902190	2	2	0	2.0	0.0	2.0	1.000000e+06	0.0	0	0	0	0	DrDoS_NetBIOS
4094982	172.16.0.5-192.168.50.1-648-45456-17	172.16.0.5	648	192.168.50.1	45456	2018-12-01 12:00:13.902193	47	2	0	47.0	0.0	47.0	4.255319e+04	0.0	0	0	0	0	DrDoS_NetBIOS
4094983	172.16.0.5-192.168.50.1-526-53287-17	172.16.0.5	526	192.168.50.1	53287	2018-12-01 12:00:13.902542	1	2	0	1.0	0.0	1.0	2.000000e+06	0.0	0	0	0	0	DrDoS_NetBIOS
4094984	172.16.0.5-192.168.50.1-527-34907-17	172.16.0.5	527	192.168.50.1	34907	2018-12-01 12:00:13.902674	1	2	0	1.0	0.0	1.0	2.000000e+06	0.0	0	0	0	0	DrDoS_NetBIOS
4094985	172.16.0.5-192.168.50.1-648-54457-17	172.16.0.5	648	192.168.50.1	54457	2018-12-01 12:00:13.902732	1	2	0	1.0	0.0	1.0	2.000000e+06	0.0	0	0	0	0	DrDoS_NetBIOS

3965133 rows × 19 columns

A continuación se han eliminado dos columnas más: “Flow ID” y “Timestamp” ya que poseen una diversidad demasiado grande para aportar valor a los posteriores modelos. Además, al no considerar la traza como una serie temporal, no es necesaria la marca de tiempo de la traza. Posteriormente se ha analizado el tipo de dato de las columnas restante y se han cambiado aquellas con valores Object o de texto en variables categóricas con un código de categoría asociado de manera que sea más fácil de manejar posteriormente.

```
#Preparacion de columnas
df.drop(['Flow ID'],axis=1,inplace=True)
df.drop(['Timestamp'],axis=1,inplace=True)

df['Source IP']=df['Source IP'].astype('category')
df['Source IP']=df['Source IP'].cat.codes

df['Source Port']=df['Source Port'].astype('category')
df['Source Port']=df['Source Port'].cat.codes

df['Destination Port']=df['Destination Port'].astype('category')
df['Destination Port']=df['Destination Port'].cat.codes

df['Destination IP']=df['Destination IP'].astype('category')
df['Destination IP']=df['Destination IP'].cat.codes

df['Label'].replace({'DrDoS_NetBIOS':1, 'BENIGN':-1}, inplace=True)

df
```

	Source IP	Source Port	Destination IP	Destination Port	Flow Duration	Total Fwd Packets	Total Backward Packets	Fwd IAT Mean	Fwd IAT Std	Fwd IAT Min	Fwd Packets/s	Bwd Packets/s	SYN Flag Count	RST Flag Count	ACK Flag Count	URG Flag Count	Label
0	7	6778	48	2331	1	2	0	1.0	0.0	1.0	2.000000e+06	0.0	0	0	0	0	1
1	7	6779	48	50166	1	2	0	1.0	0.0	1.0	2.000000e+06	0.0	0	0	0	0	1
2	7	6780	48	61530	1	2	0	1.0	0.0	1.0	2.000000e+06	0.0	0	0	0	0	1
3	7	6781	48	8926	1	2	0	1.0	0.0	1.0	2.000000e+06	0.0	0	0	0	0	1
4	7	6782	48	33036	2	2	0	2.0	0.0	2.0	1.000000e+06	0.0	0	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4094981	7	22	48	3767	2	2	0	2.0	0.0	2.0	1.000000e+06	0.0	0	0	0	0	1
4094982	7	145	48	45452	47	2	0	47.0	0.0	47.0	4.255319e+04	0.0	0	0	0	0	1
4094983	7	23	48	53283	1	2	0	1.0	0.0	1.0	2.000000e+06	0.0	0	0	0	0	1
4094984	7	24	48	34903	1	2	0	1.0	0.0	1.0	2.000000e+06	0.0	0	0	0	0	1
4094985	7	145	48	54453	1	2	0	1.0	0.0	1.0	2.000000e+06	0.0	0	0	0	0	1

3965133 rows × 17 columns

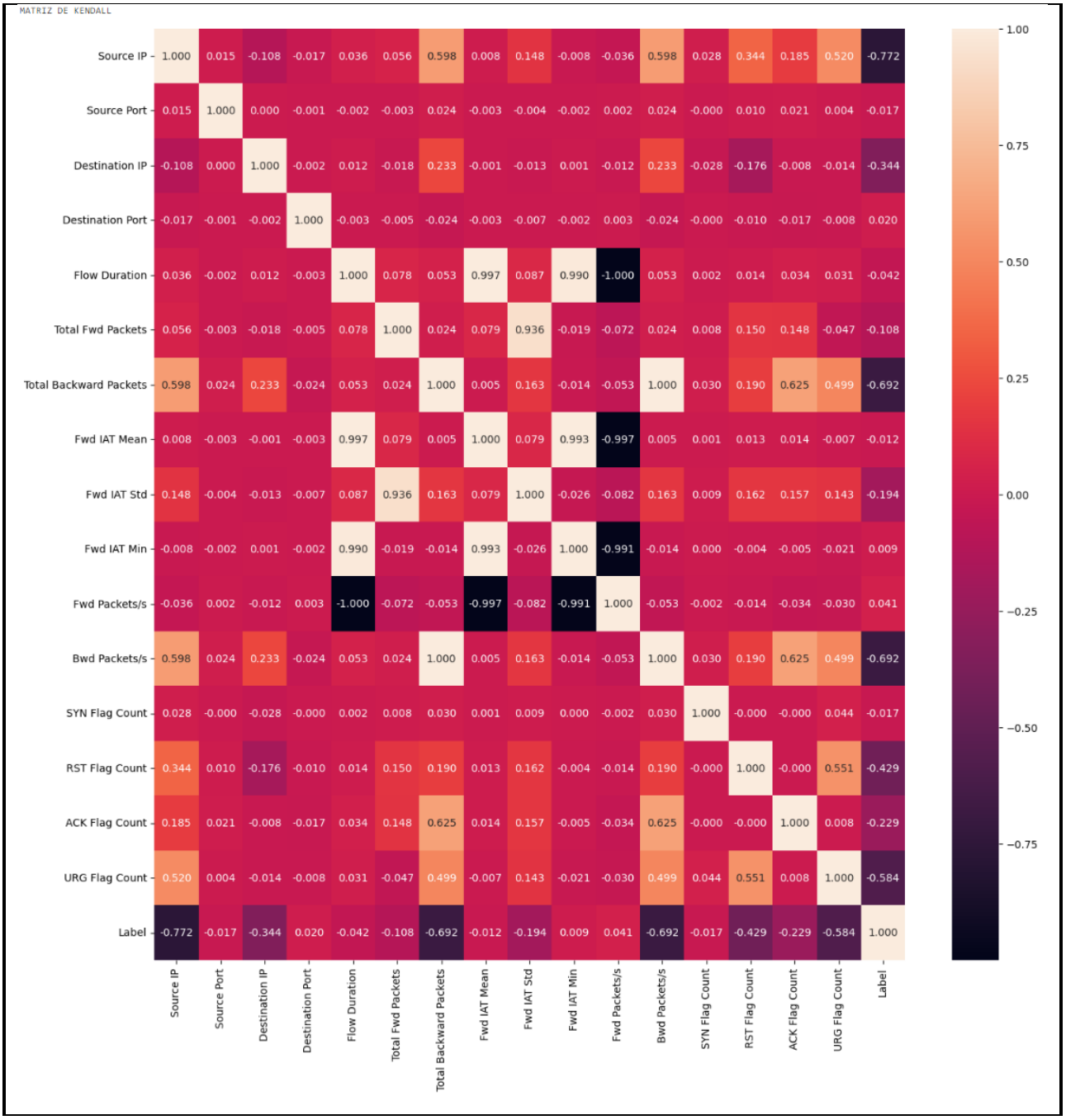
Una vez aplicadas todas las operaciones anteriores estaría listo el conjunto de datos iniciales para resolver el problema de clasificación con modelos clásicos. Sin embargo y tal, las limitaciones técnicas

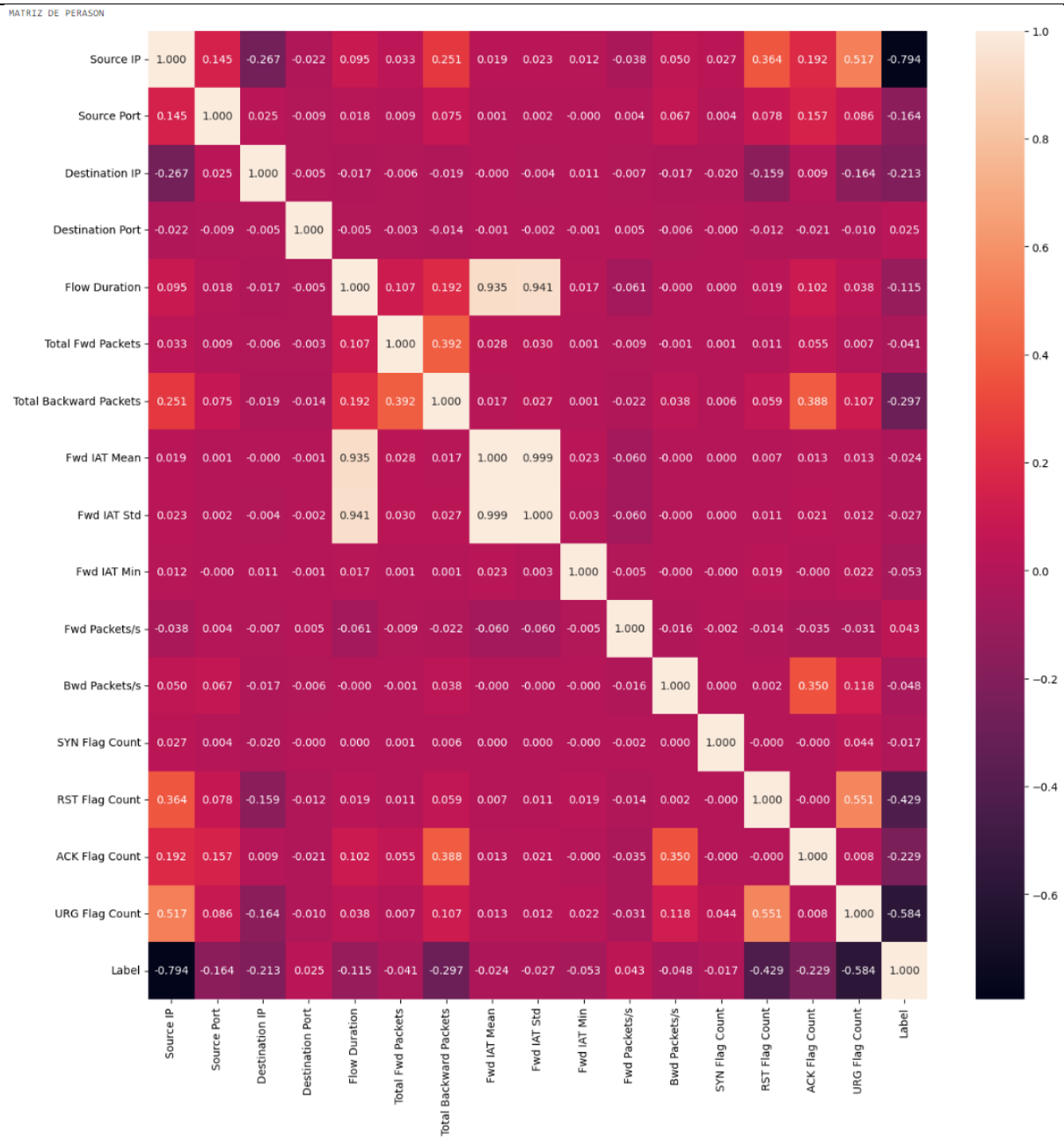
de las versiones cuánticas de los modelos no permiten operar con un número tan grande de variables en tiempos razonables. Es por ello que para conformar el dataset inicial para los modelos cuánticos de clasificación se requiere de algunos ajustes adicionales.

El primero es construir la matriz de correlaciones tanto de Pearson como la de Kendall para determinar qué variables están más relacionadas con la variable objetivo "Label". Se ha decidido construir ambas matrices de correlaciones para ver por una parte la relación lineal mediante la correlación de Pearson y por otra parte las concordancias y discordancias entre los pares de observaciones mediante la tau de Kendall.

Para ello se han empleado las librerías de Seaborn para construir las matrices de correlaciones que resulten más visuales y Matplotly para poder mostrar ambas matrices correctamente por pantalla.

```
import seaborn as sns
import matplotlib.pyplot as plt

corr_matrx = df.corr(method='kendall')

print("MATRIZ DE KENDALL")
fig, ax = plt.subplots(figsize=(16,16))
sns.heatmap(corr_matrx, annot=True, ax=ax, fmt='0.3f')
plt.show()

print("\n")

print("MATRIZ DE PERASON")
corr_matrx = df.corr()

fig, ax = plt.subplots(figsize=(16,16))
sns.heatmap(corr_matrx, annot=True, ax=ax, fmt='0.3f')
plt.show()
```


MATRIZ DE KENDALL
Source IP 1.000 0.015 -0.108 -0.017 0.036 0.056 0.598 0.008 0.148 -0.008 -0.036 0.598 0.028 0.344 0.185 0.520 -0.772
Source Port 0.015 1.000 0.000 -0.001 -0.002 -0.003 0.024 -0.003 -0.004 -0.002 0.002 0.024 -0.000 0.010 0.021 0.004 -0.017
Destination IP -0.108 0.000 1.000 -0.002 0.012 -0.018 0.233 -0.001 -0.013 0.001 -0.012 0.233 -0.028 -0.176 -0.008 -0.014 -0.344
Destination Port -0.017 -0.001 -0.002 1.000 -0.003 -0.005 -0.024 -0.003 -0.007 -0.002 0.003 -0.024 -0.000 -0.010 -0.017 -0.008 0.020
Flow Duration 0.036 -0.002 0.012 -0.003 1.000 0.078 0.053 0.997 0.087 0.990 -1.000 0.053 0.002 0.014 0.034 0.031 -0.042
Total Fwd Packets 0.056 -0.003 -0.018 -0.005 0.078 1.000 0.024 0.079 0.936 -0.019 -0.072 0.024 0.008 0.150 0.148 -0.047 -0.108
Total Backward Packets 0.598 0.024 0.233 -0.024 0.053 0.024 1.000 0.005 0.163 -0.014 -0.053 1.000 0.030 0.190 0.625 0.499 -0.692
Fwd IAT Mean 0.008 -0.003 -0.001 -0.003 0.997 0.079 0.005 1.000 0.079 0.993 -0.997 0.005 0.001 0.013 0.014 -0.007 -0.012
Fwd IAT Std 0.148 -0.004 -0.013 -0.007 0.087 0.936 0.163 0.079 1.000 -0.026 -0.082 0.163 0.009 0.162 0.157 0.143 -0.194
Fwd IAT Min -0.008 -0.002 0.001 -0.002 0.990 -0.019 -0.014 0.993 -0.026 1.000 -0.991 -0.014 0.000 -0.004 -0.005 -0.021 0.009
Fwd Packets/s -0.036 0.002 -0.012 0.003 -1.000 -0.072 -0.053 -0.997 -0.082 -0.991 1.000 -0.053 -0.002 -0.014 -0.034 -0.030 0.041
Bwd Packets/s 0.598 0.024 0.233 -0.024 0.053 0.024 1.000 0.005 0.163 -0.014 -0.053 1.000 0.030 0.190 0.625 0.499 -0.692
SYN Flag Count 0.028 -0.000 -0.028 -0.000 0.002 0.008 0.030 0.001 0.009 0.000 -0.002 0.030 1.000 -0.000 -0.000 0.044 -0.017
RST Flag Count 0.344 0.010 -0.176 -0.010 0.014 0.150 0.190 0.013 0.162 -0.004 -0.014 0.190 -0.000 1.000 -0.000 0.551 -0.429
ACK Flag Count 0.185 0.021 -0.008 -0.017 0.034 0.148 0.625 0.014 0.157 -0.005 -0.034 0.625 -0.000 -0.000 1.000 0.008 -0.229
URG Flag Count 0.520 0.004 -0.014 -0.008 0.031 -0.047 0.499 -0.007 0.143 -0.021 -0.030 0.499 0.044 0.551 0.008 1.000 -0.584
Label -0.772 -0.017 -0.344 0.020 -0.042 -0.108 -0.692 -0.012 -0.194 0.009 0.041 -0.692 -0.017 -0.429 -0.229 -0.584 1.000
Source IP
Source Port
Destination IP
Destination Port
Flow Duration
Total Fwd Packets
Total Backward Packets
Fwd IAT Mean
Fwd IAT Std
Fwd IAT Min
Fwd Packets/s
Bwd Packets/s
SYN Flag Count
RST Flag Count
ACK Flag Count
URG Flag Count
Label
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75



Con el fin de evitar correlaciones demasiado directas entre ciertas variables y la variable objetivo se han seleccionado aquellas variables que presenten relativamente poca correlación de Pearson y una ligera o moderada correlación de Kendall. De esta forma se han seleccionado únicamente las variables “Destination IP” y “Source Port”.

```
#Variable objetivo
vobj='Label'

#Definimos las variables que realmente vamos a utilizar
required_features=['Total Fwd Packets','Destination IP',vobj]


#Recortamos el dataset
df = df.loc[:,required_features]
df
```

	Total Fwd Packets	Destination IP	Label
0	2	48	1
1	2	48	1
2	2	48	1
3	2	48	1
4	2	48	1
...	...	...	...
4094981	2	48	1
4094982	2	48	1
4094983	2	48	1
4094984	2	48	1
4094985	2	48	1

3965133 rows × 3 columns

Finalmente se requiere reducir el número de instancias del problema de cara a reducir los tiempos de entrenamiento y predicción a tiempos razonables, algunos de los modelos cuánticos tienen cierta componente aleatoria que requiere de un número mínimo de repeticiones para generar la tendencia real del modelo. Teniendo en cuenta el considerable desbalance entre ambas clases, se ha decidido seleccionar el total de elementos de la clase con menos presencia y un número similar de instancias de la clase predominante.

```
#Definimos el tamaño del subconjunto de los datos a tratar
size=df[vobj].value_counts()[-1]
dataset_atk=df.loc[df['Label']==1].sample(size,random_state=0)
dataset_ben=df.loc[df['Label']==-1].sample(size,random_state=0)
dataset=pd.concat([dataset_atk, dataset_ben], axis=0)
dataset=dataset.reset_index().drop(['index'],axis=1)
dataset
```

	Total Fwd Packets	Destination IP	Label
0	2	48	1
1	2	48	1
2	2	48	1
3	2	48	1
4	2	48	1
...	...	...	...
3369	4	15	-1
3370	15	63	-1
3371	2	71	-1
3372	20	11	-1
3373	2	34	-1

3374 rows × 3 columns

Optimización

Nuestra organización no es ajena al problema de la optimización de rutas dado que lo ha abordado en varias ocasiones. Para mantenernos lo más fieles posible a la realidad, partimos de una base de datos que contiene ubicaciones reales dentro del Principado de Asturias. Es importante señalar que toda la información adjunta ha sido previamente anonimizada para garantizar la confidencialidad de los datos. En el archivo adjunto, se proporciona la longitud y latitud de cada ubicación, así como una dirección y una descripción genérica.

Aunque para las pruebas se necesita un número de puntos mucho menor, se trata de un CSV de más 1.700 líneas cuyo aspecto es similar al siguiente:

	_id	municipio_provi	poblacion	cod_postal	cod_zona	telefono	direccion	descripcion	lon	lat
0	6231b91fba05105c5696b033	ASTURIAS	VILLAPEDRE	33793	902	985472037 - 617205613 - 619577842 (DIEGO)	VILLAPEDRE	KG.PIENS VIGO/VILLAP/PI🏠/POLAV	-6,65164428634444	43,5496531891899
1	6231b91fba05105c5696b034	ASTURIAS	OTUR	33792	906	985470753	RELLON	KG.PIENS OTUR	-6,59743077517703	43,3678089941604
2	6231b91fba05105c5696b035	ASTURIAS	OTUR	33792	906	985470891 - 653025712	CALELLA	KG.PIENS OTUR	-6,58862684523012	43,5450010961423
3	6231b91fba05105c5696b036	ASTURIAS	OTUR	33792	906	985470991 - 626832157	LA CASONA	KG.PIENS OTUR	-6,59799425399774	43,5374688679064
4	6231b91fba05105c5696b037	ASTURIAS	OTUR	33792	906	985555423 - 677394250	BORONAS	KG.PIENS OTUR	-6,62104199221801	43,5278904765818
5	6231b91fba05105c5696b038	ASTURIAS	BUSMARGALI	33719	908	985979048	BUSMARGALI	KG.PIENSO BUSMARGALI	-6,65703664467469	43,5215771859461
6	6231b91fba05105c5696b039	ASTURIAS	VILLAPEDRE	33793	902		TOX	KG.PIENS VIGO/VILLAP/PI🏠/POLAV	-6,6384992751465	43,5563773007423
7	6231b91fba05105c5696b03a	ASTURIAS	OTUR	33792	901	985924185- 645678926 (MERCE)	LA ARTOSA	KG.PIENS VALLIN/ARTOSA/VIDURA	-6,61879188777164	43,5215424990179
8	6231b91fba05105c5696b03b	ASTURIAS	BUSMARGALI	33719	908	985924089	BUSMARGALI	KG.PIENSO BUSMARGALI	-6,66146765396729	43,5179634345142
9	6231b91fba05105c5696b03c	ASTURIAS	GRANAS	33717	904	985625170	GRANAS	KG.PIENSO VILLAYON	-6,67973153551247	43,4529483392061
10	6231b91fba05105c5696b03d	ASTURIAS	ONETA	33717	904	690173917-985977029	ONETA	KG.PIENSO VILLAYON	-6,66643377273863	43,4656159469744
11	6231b91fba05105c5696b03e	ASTURIAS	HERIAS	33717	904	985924472 - 697863494	HERIAS	KG.PIENSO VILLAYON	-6,67177907564087	43,43972412715
12	6231b91fba05105c5696b03f	ASTURIAS	ONETA	33717	904	985977048	ONETA	KG.PIENSO VILLAYON	-6,66762467354124	43,464486842701
13	6231b91fba05105c5696b040	ASTURIAS	VIDURAL	33718	901	656351969	VIDURAL	KG.PIENS VALLIN/ARTOSA/VIDURA	-6,63491204909667	43,5104665021046
14	6231b91fba05105c5696b041	ASTURIAS	GRANAS	33717	904	985625065	GRANAS	KG.PIENSO VILLAYON	-6,68147794343872	43,4532074394642
15	6231b91fba05105c5696b042	ASTURIAS	ONETA	33717	904	985977046 - 628286221	ONETA	KG.PIENSO VILLAYON	-6,66633721321409	43,4650397170837
16	6231b91fba05105c5696b043	ASTURIAS	BARCIA	33708	923	985979049 - 676626433	BUSMOURISCO	KG.PIENSO ZONA LEIRIELL/CABOR	-6,54943454919589	43,4668945782836
17	6231b91fba05105c5696b044	ASTURIAS	GRANAS	33717	904	985625121 - 675838003	GRANAS	KG.PIENSO VILLAYON	-6,67968593795922	43,4529288679908
18	6231b91fba05105c5696b045	ASTURIAS	NAVIA	33719	902	985472118 - 666419529	EL SEIJO	KG.PIENS VIGO/VILLAP/PI🏠/POLAV	-6,66853754526062	43,5289781314791
19	6231b91fba05105c5696b046	ASTURIAS	ILLANO	33777	921	985978098 - 658536366	SAN PEDRO DE AHIO	KG.PIENSO ZONA ILLANO/OSCOS	-6,91904081904909	43,3078246224753
20	6231b91fba05105c5696b047	ASTURIAS	TABLADO DE RIVIELLA	33784	916		TABLADO DE RIVIELLA	KG.PIENSO ZONA TINEO	-6,46298812904433	43,3758410061762
21	6231b91fba05105c5696b048	ASTURIAS	VALDEDO	33777	921	985626120	VALDEDO	KG.PIENSO ZONA ILLANO/OSCOS	-6,94580576004944	43,2551188531359
22	6231b91fba05105c5696b049	ASTURIAS	EL VIDURAL	33719	901	985977059	EL VIDURAL	KG.PIENS VALLIN/ARTOSA/VIDURA	-6,63598970740964	43,5010729041875
23	6231b91fba05105c5696b04a	ASTURIAS	LUARCA	33700	905	985640005	EL OTERO	KG.PIENS ALMU🏠A/VILLUIR	-6,54553869854738	43,5393338659296
24	6231b91fba05105c5696b04b	ASTURIAS	BUSTIELLO DE TABLADO	33874	916	985929028 - 600260315	BUSTIELLO DE TABLADO	KG.PIENSO ZONA TINEO	-6,47069048223727	43,386968850274
25	6231b91fba05105c5696b04c	ASTURIAS	EL POZON	33874	916	985909273 - 671322594	EL POZON, 1	KG.PIENSO ZONA TINEO	-6,49779204531706	43,2911030289339
26	6231b91fba05105c5696b04d	ASTURIAS	LUARCA	33787	905	985641882	BARCIA	KG.PIENS ALMU🏠A/VILLUIR	-6,50756508703307	43,5385623995204
27	6231b91fba05105c5696b04e	ASTURIAS	EL GUMIO	33729	920	985620590 - 634409588	EL GUMIO	KG.PIENSO ZONA BOAL/ELGUMIO	-6,84805680262457	43,4448377722262
28	6231b91fba05105c5696b04f	ASTURIAS	VILLAPEDRE	33793	902	985648493	TOX	KG.PIENS VIGO/VILLAP/PI🏠/POLAV	-6,63506604760744	43,5555842300106

Ahora generamos un array de objectos de tipo **Client** para almacenar los elementos leídos.

[20]:
```
import csv

# empty array
clients = []

with open('clientes_geoloc.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    for row in csv_reader:
        # Ignore column names
        if line_count != 0:
            clients.append(Client(row[0],row[8],row[9].replace(',','.'),row[10].replace(',','.')))
            # clients[line_count-1].showData()

        if line_count >= numOfClients:
            break

        line_count += 1

    print(f'Processed {line_count} lines.')
```

Posteriormente, el script infiere la dirección correspondiente a cada par longitud/latitud y la pasa a una API que calcula la distancia del recorrido por carretera entre ambas ubicaciones.

Coste de cada ruta

En cuanto al coste de cada ruta seguiremos usando la distancia, pero en lugar de hacer una recta euclidiana, utilizaremos el servicio OpenSource de OpenRoute (https://openrouteservice.org/) para que el cálculo sea lo más realista posible.

Para ello, definimos una función que llama a dicho servicio y retorna la información sobre tiempo y distancia:

[21]:
```
import requests

class OpenRouteServiceManager:

    def get_distance_matrix_public(locations, token="5b3ce3597851110001cf624830f3064d1e8549eea963fbae1788556e"):

        chunks = list(locations)

        distance_matrix = np.full([len(chunks), len(chunks)], None)

        time_matrix = np.full([len(chunks), len(chunks)], None)

        url = "https://api.openrouteservice.org/v2/matrix/driving-car"

        body = {'locations': locations,
                "metrics": ["distance", "duration"]
               }

        headers = {
                'Accept': 'application/json, application/geo+json, application/gpx+xml, img/png; charset=utf-8',
                'Authorization': token,
                'Content-Type': 'application/json; charset=utf-8'
               }

        res = requests.post(url, json=body, headers=headers).json()

        distance_matrix = res['distances']
        time_matrix = res['durations']

        return distance_matrix, time_matrix
```

Simulación

Los conjuntos de datos para el problema de simulación deben basarse, por motivos de carga computacional, en moléculas relativamente sencillas, es decir, que tienen un número limitado de átomos. Esto se debe a que, tanto la documentación consultada como las pruebas iniciales, demuestran que la complejidad del problema aumenta exponencialmente con el tamaño de la molécula en términos de número de átomos. Los cuellos de botella ocurren principalmente en la CPU para un número bajo de moléculas, y se estima que el cuello de botella migrará a ser principalmente en la memoria a medida que el tamaño del problema aumente.

Concretamente, las tres moléculas escogidas para la implementación del problema de simulación son las siguientes. Se han escogido estos ejemplos porque demuestran el problema para un número creciente y limitado de átomos, y además porque los resultados teóricos esperados son bien conocidos, lo que facilita la validación:

- Molécula de **dihidrógeno** (H_2).
- Molécula de **agua** (H_2O).
- Molécula de **metano** (CH_4). Es importante destacar que el número de átomos de esta molécula ya la convierte en un problema de computación significativo.

Aunque las características estructurales de estas moléculas son fácilmente accesibles, las librerías software utilizadas requieren un formato específico. A continuación, se presentan las representaciones de dichas estructuras utilizadas en los experimentos, empleando los constructos propios de cada librería (Qiskit, PennyLane, PSI4).

Representación de dihidrógeno (H_2) en Qiskit Nature

```
from qiskit_nature.drivers import Molecule
from qiskit_nature.drivers.second_quantization import (
   ElectronicStructureMoleculeDriver,
   ElectronicStructureDriverType,
)

mol = Molecule(geometry=[["H", [0.0, 0.0, -0.37]], ["H", [0.0, 0.0, 0.37]]])
```

```
driver = ElectronicStructureMoleculeDriver(
   mol, basis="sto3g", driver_type=ElectronicStructureDriverType.PYSCF
)
```

Representación de dihidrógeno (H_2) en PennyLane

```
import pennylane as qml
from pennylane import numpy as np

symbols = ["H", "H"]
coordinates = np.array([0.0, 0.0, -0.6991986158, 0.0, 0.0, 0.6991986158])
H, qubits = qml.qchem.molecular_hamiltonian(symbols, coordinates)
```

Representación de agua (H_2O) en Qiskit Nature

```
from qiskit_nature.drivers import Molecule
from qiskit_nature.drivers.second_quantization import (
   ElectronicStructureMoleculeDriver,
   ElectronicStructureDriverType,
)

mol = Molecule(
   geometry=[
      ["O", [-0.0402629934, 0.0520003765, 0.0000000000]],
      ["H", [0.9197370066, 0.0520003765, 0.0000000000]],
      ["H", [-0.2806277973, -0.8774213582, 0.0000000000]],
   ]
)


driver = ElectronicStructureMoleculeDriver(
   mol, basis="sto3g", driver_type=ElectronicStructureDriverType.PYSCF
)
```

Representación de agua (H_2O) en Qiskit Nature basada en el driver de interacción con PySCF

```
from qiskit_nature.units import DistanceUnit
from qiskit_nature.second_q.drivers import PySCFDriver

driver = PySCFDriver(
   atom="O -0.0402629934 0.0520003765 0.0000000000; H 0.9197370066 0.0520003765
0.0000000000; H -0.2806277973 -0.8774213582 0.0000000000",
   basis="sto3g",
   charge=0,
   spin=0,
   unit=DistanceUnit.ANGSTROM,
)
```

Representación de agua (H_2O) en PennyLane

```
import pennylane as qml
```

```
from pennylane import numpy as np

symbols = ["O", "H", "H"]

geometry = (
    np.array(
        [
            -0.0402629934,
            0.0520003765,
            0.0000000000,
            0.9197370066,
            0.0520003765,
            0.0000000000,
            -0.2806277973,
            -0.8774213582,
            0.0000000000,
        ]
    )
    * 1.8897259886
)

electrons = 2
charge = 0

# Electronic Hamiltonian: the operator that represents the energy of the system
H, qubits = qml.qchem.molecular_hamiltonian(symbols, geometry, charge=charge)

# Hartree-Fock state: the initial state for quantum chemistry simulations
hf_state = qml.qchem.hf_state(electrons, qubits)

# Generate single and double excitations
singles, doubles = qml.qchem.excitations(electrons, qubits)
```

Representación de agua (H_2O) en PSI4

```
import psi4

h2o = psi4.geometry(
    """
O
H 1 0.96
H 1 0.96 2 104.5
"""
)

psi4.set_options({"scf_type": "pk", "basis": "3-21G", "reference": "rhf"})
```

Representación de metano (CH_4) en Qiskit Nature

```
from qiskit_nature.drivers import Molecule
from qiskit_nature.drivers.second_quantization import (
    ElectronicStructureMoleculeDriver,
```

```
    ElectronicStructureDriverType,
)

mol = Molecule(
    geometry=[
        ["c", [0.000000, 0.000000, 0.000000]],
        ["H", [0.000000, 0.000000, 1.089000]],
        ["H", [1.089000, 0.000000, 0.000000]],
        ["H", [-0.544500, -0.943102, 0.000000]],
        ["H", [-0.544500, 0.943102, 0.000000]],
    ]
)


driver = ElectronicStructureMoleculeDriver(
    mol, basis="sto3g", driver_type=ElectronicStructureDriverType.PYSCF
)
```

Representación de metano (CH_4) en Qiskit Nature basada en el driver de interacción con PySCF

```
from qiskit_nature.units import DistanceUnit
from qiskit_nature.second_q.drivers import PySCFDriver

driver = PySCFDriver(
    atom=""" C     0.000000     0.000000     0.000000;
 H     0.000000     0.000000     1.089000;
 H     1.089000     0.000000     0.000000;
 H    -0.544500    -0.943102     0.000000;
 H    -0.544500     0.943102     0.000000""",
    basis="sto3g",
    charge=0,
    spin=0,
    unit=DistanceUnit.ANGSTROM,
)
```

Representación de metano (CH_4) en PSI4

```
import psi4

methane = psi4.geometry(
    """
 C     0.000000     0.000000     0.000000
 H     0.000000     0.000000     1.089000
 H     1.089000     0.000000     0.000000
 H    -0.544500    -0.943102     0.000000
 H    -0.544500     0.943102     0.000000
units angstrom
"""
)

psi4.set_options({"scf_type": "pk", "basis": "3-21G", "reference": "rhf"})
```

T5.3: Pruebas de laboratorio

Las actividades realizadas en esta tarea se muestran a continuación.

Clasificación
A la hora de abordar el problema de clasificación de ciberataques sobre el conjunto de datos definido se ha obtenido en primer lugar el mejor modelo tanto para la versión clásica como para la cuántica y posteriormente se han obtenido sus métricas finales.

Modelo clásico

Para determinar el mejor modelo clásico se ha realizado un proceso de grid-seach para determinar el modelo que mejor se ajusta a los datos del problema. Para ello se ha hecho una selección de modelos clásicos con sus respectivos hiperparámetros y se ha probado sobre un subconjunto de los datos empleado para entrenar que supone el 80% de los datos empleando cross-validation con 10 capas y también sobre el 20% restante de los datos que se emplean como datos de test.

	Modelo	accuracy_cv	precision_cv	recall_cv	f1_cv	roc_auc_cv	accuracy_test	precision_test	recall_test	f1_test	roc_auc_test
0	RL	0.998744	0.999998	0.998745	0.999371	0.998867	0.998768	1.000000	0.998767	0.999383	0.999384
1	SVM	0.999598	0.999600	0.999998	0.999799	0.998512	0.999584	0.999586	0.999998	0.999792	0.502931
2	SGD	0.999758	0.999984	0.999774	0.999879	0.994005	0.000416	0.000000	0.000000	0.000000	0.500000
3	DT	0.983062	0.999953	0.983102	0.991456	0.956092	0.983105	0.999959	0.983138	0.991477	0.943182
4	RF	0.914569	0.999950	0.914579	0.955361	0.913840	0.914482	0.999957	0.914486	0.955313	0.910322
5	XGB_tree	0.999136	0.999998	0.999137	0.999567	0.997579	0.999214	1.000000	0.999213	0.999607	0.999607
6	XGB_linear	0.832092	0.999953	0.832061	0.908314	0.946709	0.832611	0.999960	0.832575	0.908623	0.876698

Los resultados obtenidos reflejan que varios modelos tienen un rendimiento similar tanto en los datos con cross-validation como en test por lo que se ha hecho una prueba adicional reentrenando los mejores modelos y prediciendo los datos de conjunto de test para medir sus respectivos tiempos de ejecución.

Modelo	Tiempo entrenamiento	Tiempo prediccion
LogisticRegression(C...	35.379982924461366	0.03551270961761475
DecisionTreeClassifie...	1.190125322341919	0.04075193405151367
RandomForestClassif...	20.757201313972473	1.3253301858901978
XGBClassifier(base_s...	60.0190853357315	0.06394777297973633

De entre los candidatos finales, el modelo que mejor rendimiento ha obtenido es el modelo Decision Tree con tiempos de entrenamiento y predicción considerablemente mejores que los demás y dado que, como se ha comentado anteriormente, los cuatro modelos tienen métricas de predicción similares, este será el modelo final a comparar con la versión cuántica.

Modelo cuántico

Los modelos cuánticos empleados en estas pruebas son los definidos en tareas previas. Cada uno de estos se ha probado sobre la versión cuántica del conjunto de datos y se han medido tanto sus métricas de rendimiento del modelo como los tiempos de entrenamiento y de predicción. Cada uno de los modelos se ha ejecutado un cierto número de veces y se han tomado valores medios de los resultados para evitar la componente del estado del sistema operativo en los tiempos de ejecución ya que no se han podido ejecutar de manera totalmente aislada. Por limitaciones técnicas y dada las características de cada modelo se han ejecutado las siguientes pruebas:

- Modelo Variacional con 2 qubits y 1.000 repeticiones debido a su componente aleatorio.
- Modelo Kernel-based con 10 repeticiones debido a los tiempos de ejecución que se presentarán posteriormente y que no tiene componente aleatorio.
- Modelo Quantum K-Neighbours con 100 repeticiones ya que posee un ligero componente aleatorio.

Los resultados de rendimiento de los 3 modelos implementados son los siguientes:

	Modelo	Repeticiones	Accuracy	Precision	Recall	F1
0	Variational	1000	0.75	0.71	0.83	0.79
1	Kernel	10	0.90	0.90	0.89	0.90
2	QKNN	100	0.82	0.81	0.85	0.82

En general los 3 modelos se desempeñan bien con el conjunto de datos elegido destacando el modelo Kernel con respecto a su capacidad de identificar ambas clases con mayor claridad que el resto de modelos.

En cuanto a los tiempos de ejecución tanto en el entrenamiento como la inferencia se han obtenido los siguientes resultados:

	Modelo	Tiempo entrenamiento medio	Tiempo prediccion medio
0	Variational	32.34	3.89
1	Kernel	371.97	80.64
2	QKNN	28.90	30.27

Se puede observar que el modelo Kernel requiere de tiempos significativamente mayores para su ejecución tanto en train como en inferencia mientras que el modelo QKNN es ligeramente mejor que el modelo variacional en train pero considerablemente peor en tiempos de inferencia mostrando a priori el modelo variacional como la opción más balanceada en cuanto a tiempos de ejecución y rendimiento del modelo.

Optimización

Análisis y preparación del conjunto de datos

Los datos usados para la resolución del subproblema de optimización pueden diferenciarse claramente en dos grandes bloques:
En la primera fase, para la resolución del problema se parte de una matriz de distancia generada con datos aleatorios.
En la segunda fase, se ha optado por utilizar datos reales de producción para que la aplicación de los algoritmos sean más realistas. En la base de datos origen los puntos estaban geolocalizados mediante longitud y latitud. Para usar un contexto aún más realista, para los puntos seleccionados se han calculado las distancias reales (en vez de utilizar líneas rectas).
Para obtener dichas rutas usamos Openrouteservice API, que provee algoritmos de enrutamiento que trabajan sobre datos geográficos libres de OpenStreetMap. Es un servicio libre basado en web que puede ser accedido mediante un complemento QGIS. Aunque el servicio es libre, requiere registrarse y obtener una llave API.
A continuación se expone el código para generar la matriz aleatoria de la primera fase.
Primero inicializamos las variables para las distintas pruebas:

```
# Initialize the problem by defining the parameters
n = 4  # number of nodes + depot (n+1)
K = 2  # number of vehicles
```

Definimos una clase de inicialización que coloca aleatoriamente los nodos en un plano 2D y calcula la distancia entre ellos.

```
# Get the data
class Initializer:
    def __init__(self, n):
        self.n = n

    def generate_instance(self):

        n = self.n

        # np.random.seed(33)
        np.random.seed(1543)

        xc = (np.random.rand(n) - 0.5) * 10
        yc = (np.random.rand(n) - 0.5) * 10

        instance = np.zeros([n, n])
        for ii in range(0, n):
            for jj in range(ii + 1, n):
                instance[ii, jj] = (xc[ii] - xc[jj]) ** 2 + (yc[ii] - yc[jj]) ** 2
                instance[jj, ii] = instance[ii, jj]

        return xc, yc, instance
```

El resultado de llamar al método generate_instance devolverá:

```
print(instance)
```

```
[[ 0.         36.84023052  5.061353   30.63150414]
 [36.84023052  0.         24.55322904 63.21893593]
 [ 5.061353   24.55322904  0.         15.49719877]
 [30.63150414 63.21893593 15.49719877  0.        ]]
```

Si pasamos a los datos reales, el proceso se complica. Primero, creamos una clase Client para almacenar los datos de cada cliente (que tendrá un Id, una descripción del cliente, y su longitud y latitud):

```
class Client:
    def __init__(self, id, description, lon, lat):
        self.id = id
        self.description = description
        self.lon = lon
        self.lat = lat

    def showData(self):
        print(str(self.id) + " - " + str(self.description) + "   - lon: " + str(self.lon) + "   - lat: " + str(self.lat))

    def toJson(self):
        return json.dumps(self)
```

Ahora generamos un array de objetos de tipo Client para almacenar los elementos leídos del fichero CSV que contiene los datos reales de producción.

```
# empty array
clients = []

with open('clientes_geoloc.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    for row in csv_reader:
        # Ignore column names
        if line_count != 0:
            clients.append(Client(row[0],row[8],row[9].replace(',','.'),row[10].replace(',','.')))
            # clients[line_count-1].showData()

        if line_count >= numOfClients:
            break

        line_count += 1

    print(f'Processed {line_count} lines.')
```

A continuación, tal y como hemos comentado, construimos una nueva matriz de distancias llamando al servicio de OpenRouteService para obtener tanto la distancia real como el cálculo del coste temporal entre puntos:

```
class OpenRouteServiceManager:

    def get_distance_matrix_public(locations, token="[redacted]"):

        chunks = list(locations)

        distance_matrix = np.full([len(chunks), len(chunks)], None)

        time_matrix = np.full([len(chunks), len(chunks)], None)

        url = "https://api.openrouteservice.org/v2/matrix/driving-car"

        body = {'locations': locations,
                "metrics": ["distance", "duration"]
                }

        headers = {
                    'Accept': 'application/json, application/geo+json, application/gpx+xml, img/png; charset=utf-8',
                    'Authorization': token,
                    'Content-Type': 'application/json; charset=utf-8'
                }

        res = requests.post(url, json=body, headers=headers).json()

        distance_matrix = res['distances']
        time_matrix = res['durations']

        return distance_matrix, time_matrix
```

Resultados comparativos

Una vez llevadas a cabo todas las operaciones en la matriz aleatoria y en la real, los resultados son los siguientes:

Cuatro nodos (n = 4)

TIEMPO DE PROCESADOR (CPU time):
- Computación clásica datos aleatorios: 455 ms
- Computación cuántica datos aleatorios: 14 s
- Computación clásica datos reales: 9.31 ms
- Computación cuántica datos reales: 13.2 s

TIEMPO REAL (WALL time):
- Computación clásica datos aleatorios: 77.9 ms
- Computación cuántica datos aleatorios: 14 s
- Computación clásica datos reales: 7.48 ms
- Computación cuántica datos reales: 13.3 s

Cinco nodos (n = 5)

TIEMPO DE PROCESADOR (CPU time):
- Computación clásica datos aleatorios: 92.6 ms
- Computación cuántica datos aleatorios: 20 min 27s
- Computación clásica datos reales: 264 ms
- Computación cuántica datos reales: 20 min 23s

TIEMPO REAL (WALL time):
- Computación clásica datos aleatorios: 42.4 ms
- Computación cuántica datos aleatorios: 20 min 27s
- Computación clásica datos reales: 36.9 ms
- Computación cuántica datos reales: 20 min 24s

Cabe aclarar, entre otras cosas, el motivo de que el tiempo real pueda ser menor que el tiempo de ejecución. Para explicarlo, tomemos por ejemplo que un cálculo requiere dos segundos de tiempo de procesador. Dos procesadores pueden (idealmente) completarlo en un segundo. Por lo tanto, un sistema de dos procesadores tiene dos segundos de CPU por cada segundo de tiempo real (WALL time). Incluso si no se usa procesamiento paralelo explícitamente, la biblioteca de Python o el propio sistema operativo pueden usar múltiples procesadores de manera interna para realizar el trabajo para su proceso.

Resultados Optimización

En el ámbito de las Ciencias de la Computación, la optimización de rutas constituye uno de los problemas más célebres y estudiados. Comúnmente conocido como "El Problema del Viajante" (TSP por sus siglas en inglés, Travelling Salesman Problem), se trata de un problema NP-Completo que involucra un mapa con una serie de paradas que deben recorrerse con el menor costo posible. El costo puede estar relacionado con la distancia, el consumo o el tiempo (entre otros), dependiendo de la aproximación al problema.

Para realizar el cálculo de las rutas, es necesario representar el mapa real como un grafo donde los nodos corresponden a las paradas y los costos se reflejan en los arcos que las conectan. Dado que en la realidad el costo puede variar al ir de un punto A a un punto B en comparación con el recorrido inverso, el grafo es dirigido. Esto puede deberse a diferencias en la carretera o en la ruta seleccionada.

En esta sección plasmamos los resultados obtenidos mediante el algoritmo clásico y el cuántico en los experimentos que se vienen explicando. También se incluye una representación gráfica en forma de grafo de la ruta resultante en cada uno. Esta es la manera más clásica de representar estas soluciones.

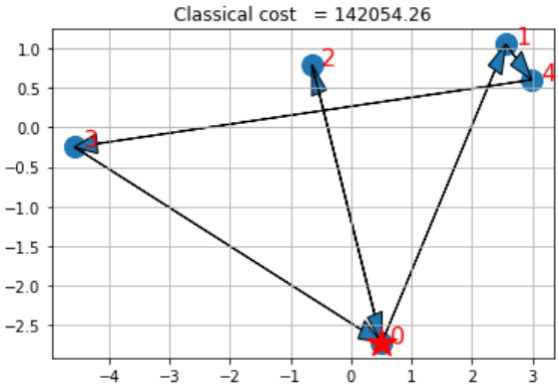


Figura 1. Representación gráfica del algoritmo clásico junto con su coste temporal (en ms).

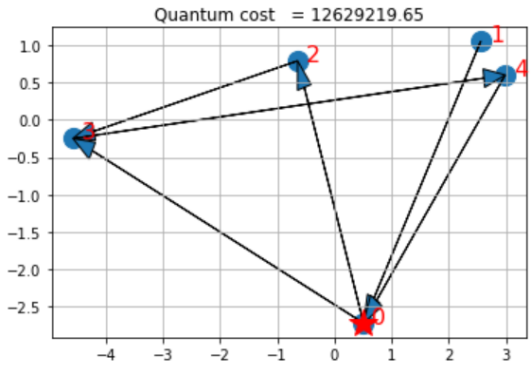


Figura 2. Representación gráfica del algoritmo cuántico junto con su coste temporal (en ms).

Conclusiones Optimización

Los resultados obtenidos ponen de manifiesto la brecha existente entre los planteamientos cuánticos desarrollados hasta la fecha y la tecnología disponible hoy en día. IBM es una empresa que suministra periódicamente procesadores cuánticos superconductores cada vez más potentes, junto con avances cruciales en orquestación de sistemas cuánticos clásicos y software. Pese a todo, sus algoritmos de última generación, disponiendo de tecnología impensable treinta años atrás, se queda atrás en comparación con los algoritmos de computación clásica (que llevan muchísimo más tiempo siendo optimizados para la tecnología actual).

A nivel de escalado, vemos que el salto de 4 a 5 vértices es tan exponencial para los algoritmos cuánticos que se llega a pasar de 14 segundos a 20 minutos. Hemos intentado plasmar los resultados para 6 vértices para afinar más la función de tiempo, pero se ha disparado tanto que ha sido inviable. Por poner en contexto lo que esto representa, cabe tener en cuenta que en producción se pueden necesitar estos cálculos para más de 100 nodos, lo cual queda totalmente descartado para los algoritmos cuánticos actuales.

¿Quiere esto decir que los resultados no sean prometedores? La respuesta es no. Pese a que en la actualidad los algoritmos clásicos sean mejores en lo que a coste temporal se refiere, hay que resaltar que los algoritmos han mejorado mucho en los últimos años, y gracias al esfuerzo de empresas como IBM siguen haciéndolo día a día. De la mano de una mejora en el hardware, estos pequeños avances son los que van abriendo camino para llegar a una computación cuántica real y aplicable.

Simulación

Como se comenta en otras actividades, en el caso de uso de simulación química se han planteado tres moléculas como conjuntos de datos de partida, cuyos resultados se presentan en esta sección:

- **Molécula de dihidrógeno** (H_2). Esta molécula ha sido explorada en el documento E3.4 "Validación de algoritmos de computación cuántica". Su objetivo ha sido actuar como el problema más simple posible, pero realista, para la validación de los algoritmos.
- **Molécula de agua** (H_2O). Esta molécula es ligeramente más compleja que la molécula de H_2 y, aún así, representa un desafío computacional significativo. Se ha logrado completar la optimización electrónica de la molécula de agua, obteniendo valores energéticos similares a los del caso clásico, pero con un coste computacional mucho más alto. Esto hacía prohibitivo intentar una optimización geométrica.
- **Molécula de metano** (CH_4). En el caso de la molécula de metano, tanto el tiempo de cálculo empleado como diversos problemas con los softwares empleados han hecho inviable su realización.

Acerca de los resultados obtenidos con computación clásica

En contraste con los problemas de optimización y clasificación, la simulación se presenta como un caso tan complejo computacionalmente para el ámbito de la computación cuántica, que obtener resultados mediante el método clásico resulta significativamente más sencillo. De tal manera que las conclusiones valiosas y lecciones aprendidas derivan más de las dificultades y retos enfrentados al implementar el problema con métodos cuánticos que de la comparación directa de resultados entre los enfoques clásicos y cuánticos para evaluar la eficacia de las técnicas cuánticas.

Para más detalles, los scripts que implementan el problema para las moléculas de metano y agua utilizando métodos de computación clásica se pueden encontrar en el documento E2.3 "Implementación de algoritmos de computación clásica". A continuación se incluyen los resultados generados por estos scripts en una máquina de desarrollo personal, es decir, en una infraestructura significativamente menos capaz que el servidor de altas prestaciones en el que se ejecutan los procesos basados en computación cuántica:

Metano funcional no-híbrido
Optimizer: Optimization complete! tiempo empleado (usando process_time): 12.001305953 segundos {'ps_ene_0': [-40.22222448001871], 'ps_ene_f': [-40.26207874539899], 'ps_HOMO': 0.11854794498344086, 'ps_LUMO': 0.1649975189571585, 'HOMO-LUMO_gap': 0.04644957397371764}

Agua funcional no-híbrido

Optimizer: Optimization complete!
tiempo empleado (usando process_time): 3.622535977 segundos
{'ps_ene_0': [-75.94726886708371], 'ps_ene_f': [-75.95124831103305], 'ps_HOMO': 0.05287157144619063, 'ps_LUMO': 0.1376073700027705, 'HOMO-LUMO_gap': 0.08473579855657988}

Metano funcional híbrido

Optimizer: Optimization complete!
tiempo empleado (usando process_time): 17.319405948 segundos
{'ps_ene_0': [-40.260624365015985], 'ps_ene_f': [-40.30159457956264], 'ps_HOMO': 0.14591079844713983, 'ps_LUMO': 0.19400004517353092, 'HOMO-LUMO_gap': 0.04808924672639109}

Agua funcional híbrido

Optimizer: Optimization complete!
tiempo empleado (usando process_time): 4.1913188329999995 segundos
{'ps_ene_0': [-75.97180560136869], 'ps_ene_f': [-75.97396484893277], 'ps_HOMO': 0.08727968968293633, 'ps_LUMO': 0.1767073862569143, 'HOMO-LUMO_gap': 0.08942769657397796}

Determinación de la energía del ground state de una molécula de agua

Además de las pruebas con moléculas simples como la de dihidrógeno, se han realizado experimentos para intentar comprobar si métodos como el VQE son útiles para el estudio de moléculas más complejas. Como primer paso en esta dirección, se ha considerado la molécula de agua (H_2O).

La geometría considerada para la molécula ha sido la siguiente:

```
Molecule:
        Multiplicity: 1
        Charge: 0
        Unit: Angstrom
        Geometry:
                O       [-0.0402629934, 0.0520003765, 0.0]
                H       [0.9197370066, 0.0520003765, 0.0]
                H       [-0.2806277973, -0.8774213582, 0.0]
```

Experimentos con Qiskit Nature 0.4.5

Los primeros experimentos se llevaron a cabo con la versión 0.4.5 de Qiskit Nature. En primer lugar, se obtuvo el hamiltoniano fermiónico. Este hamiltoniano resultó ser notablemente más grande que en el caso de la molécula de dihidrógeno. En concreto, se obtuvo un hamiltoniano de 5.230 términos, los primeros de los cuales se muestran a continuación:

```
-32.700062146730545 * ( +_0 -_0 )
+ 0.558237421847923 * ( +_0 -_1 )
+ 2.2740641743906417e-11 * ( +_0 -_2 )
+ 0.2348049250514537 * ( +_0 -_3 )
```

```
+ 0.30387689555384473 * ( +_0 -_5 )
+ -8.07795390305 ...
```

A continuación, se utilizó la transformación de Jordan-Wigner para obtener un hamiltoniano expresado en función de productos tensoriales de matrices de Pauli. El hamiltoniano obtenido consta de más de 1.000 términos en 14 qubits. Los primeros términos de su expresión son los siguientes:

```
-55.59940937012721 * IIIIIIIIIIIII
+ 12.413278779498107 * IIIIIIIIIIIIZ
+ 0.12507471481279436 * IIIIIIIIIIIYY
+ 0.12507471481279436 * IIIIIIIIIIIXX
+ 0.042680203537456544 * IIIIIIIIIYZZY
+ 0.042680203537456544 * IIIIIIIIIXZZX
+ 0.07248175205286048 * IIIIIIIYZZZZY
+ 0.07248175205286048 * IIIIIIIXZZZZX
+ 1.655898613192636 * IIIIIIIIIIIZI ...
```

Después de esto, se procedió a ejecutar el algoritmo VQE. Como forma variacional se eligió UCCSD, en vista de los resultados anómalos obtenidos con EfficientSU2 en la determinación del perfil de disociación de la molécula de dihidrógeno (véase la sección anterior).

Es de destacar que la ejecución de este algoritmo fue extremadamente lenta. En total, fueron necesarios más de 4 días de cálculo en el simulador cuántico para obtener un resultado. Esto contrasta con los tiempos de ejecución necesarios con métodos tradicionales, que son de apenas unos pocos minutos. Las razones que explican la ineficiencia del proceso de ejecución de VQE incluyen las siguientes:

- Dado el elevado número de términos del hamiltoniano, es necesario ejecutar muchos circuitos independientes para poder estimar la energía de un cierto estado. Cada uno de estos circuitos añade una serie de puertas de rotación con el objeto de realizar un cambio de base y medir el estado en la base de autovalores correspondientes al producto tensorial de matrices de Pauli que se esté considerando. Este incremento de circuitos a ejecutar quizá podría reducirse intentando hacer alguna poda en la expresión del hamiltoniano. Por ejemplo, términos que contribuyan menos de un cierto umbral de energía podrían eliminarse para simplificar la expresión del hamiltoniano a tratar.
- Los circuitos se están ejecutando en un simulador cuántico. Evidentemente, esto no es óptimo. Métodos como el VQE están pensados para delegar la ejecución de los circuitos a un procesador cuántico, lo que debería acelerar considerablemente el proceso (al menos, cuando la tecnología cuántica se encuentre suficientemente madura).
- El número de qubits considerado en el caso de la molécula de agua es de 14. Puesto que la dificultad de simular los circuitos crece exponencialmente con el número de qubits, este caso es notablemente más difícil que el de la molécula de dihidrógeno, que solo necesitaba 4 qubits. Una forma de paliar este incremento de complejidad podría ser intentar explotar alguna simetría en la geometría de la molécula para reducir el número de qubits necesarios.

En cualquier caso, finalmente se pudieron obtener resultados para el caso de la molécula de agua, que son los que se muestran a continuación:

```
=== GROUND STATE ENERGY ===

* Electronic ground state energy (Hartree): -84.181245478728
  - computed part:      -84.181245478728
~ Nuclear repulsion energy (Hartree): 9.168193301034
> Total ground state energy (Hartree): -75.013052177693

=== MEASURED OBSERVABLES ===

  0:  # Particles: 10.000 S: 0.000 S^2: 0.000 M: 0.000

=== DIPOLE MOMENTS ===

~ Nuclear dipole moment (a.u.): [0.59905313  -0.77368783  0.0]

  0:
  * Electronic dipole moment (a.u.): [0.21142191  -0.27305013  -0.0000007]
    - computed part:      [0.21142191  -0.27305013  -0.0000007]
  > Dipole moment (a.u.): [0.38763122  -0.5006377  0.0000007]  Total: 0.63316354
                 (debye): [0.98526019  -1.272494  0.00000179]  Total: 1.60934105
```

Estos valores fueron comprobados con la ejecución de métodos clásicos con la misma configuración, que arrojaron resultados prácticamente idénticos. Por tanto, el método es efectivo pero no eficiente (al menos, en su simulación en ordenadores clásicos y sin realizar simplificaciones encaminadas a reducir el tiempo de cómputo).

Experimentos con Qiskit Nature 0.5.0

Como comprobación adicional de los resultados obtenidos y con el objetivo de intentar reducir el tiempo de ejecución en la determinación del estado fundamental de la molécula de agua, se ejecutó el algoritmo VQE con la implementación de la versión 0.5.0 de la librería Qiskit Nature.

De nuevo, los tiempos de ejecución fueron superiores a los cuatro días y los resultados obtenidos fueron consistentes con los hallados con anterioridad, tanto con métodos cuánticos como con métodos clásicos. En concreto, fueron los siguientes:

```
=== GROUND STATE ENERGY ===

* Electronic ground state energy (Hartree): -84.181348002534
  - computed part:      -84.181348002534
~ Nuclear repulsion energy (Hartree): 9.168193301034
> Total ground state energy (Hartree): -75.0131547015

=== MEASURED OBSERVABLES ===

  0:  # Particles: 10.000 S: 0.000 S^2: 0.000 M: 0.000
```

```
=== DIPOLE MOMENTS ===

~ Nuclear dipole moment (a.u.): [0.59905313  -0.77368783  0.0]

 0:
 * Electronic dipole moment (a.u.): [0.21076797  -0.27221061  0.0]
   - computed part:      [0.21076797  -0.27221061  0.0]
 > Dipole moment (a.u.): [0.38828516  -0.50147722  0.0]  Total: 0.6342277
             (debye): [0.98692234  -1.27462784  0.0]  Total: 1.61204585
```

Experimentos con PennyLane

Para los experimentos con la molécula de agua y PennyLane, se transformó en primer lugar la geometría de la molécula a unidades atómicas:

```
symbols  = ['O', 'H', 'H']
geometry = np.array([-0.0402629934,    0.0520003765,    0.0000000000, 0.9197370066,
0.0520003765,    0.0000000000, -0.2806277973,   -0.8774213582,    0.0000000000])*1.8897259886
```

De nuevo, se obtuvo un hamiltoniano con cientos de términos, los primeros de los cuales se muestran a continuación:

```
  (-46.431215922959225) [I0]
+ (0.7796643817170361) [Z10]
+ (0.7796643817170363) [Z11]
+ (0.8068257131603902) [Z12]
+ (0.8068257131603902) [Z13]
+ (1.2088009298084208) [Z4]
+ (1.208800929808421) [Z5]
+ (1.314450110886486) [Z7]
+ (1.3144501108864863) [Z6]
+ (1.3705483218046592) [Z8]
+ (1.3705483218046595) [Z9]
+ (1.6558987877592897) [Z2]
+ (1.65589878775929) [Z3]
+ (12.413278761364191) [Z1]
+ (12.413278761364193) [Z0]
+ (-0.10420529012775617) [Y0 Y2]
+ (-0.10420529012775617) [X0 X2]
...
```

Para la implementación del algoritmo VQE, se usó el descenso del gradiente como minimizador, aprovechando la posibilidad que ofrece PennyLane de utilizar diferenciación simbólica a la hora de calcular las derivadas de las funciones de los valores esperados de los estados cuánticos. Durante este proceso, se encontraron numerosos problemas:

- El hamiltoniano obtenido variaba aleatoriamente de una ejecución a otra. Se trataba de diferencias pequeñas, pero apreciables a la hora de aplicar el algoritmo. Esto fue reportado a

Xanadu, la empresa desarrolladora de PennyLane. La solución adoptada fue establecer una semilla para la librería numpy antes de calcular el hamiltoniano.

- La minimización convergía a valores distintos dependiendo del punto de inicio que se tomara para los parámetros. Además, los valores obtenidos se encontraban lejos de los obtenidos con Qiskit. Se intentó validar el hamiltoniano con el que se estaba trabajando, pero PennyLane no ofrece métodos para resolver el problema de forma clásica. Se consultó a Xanadu, sin obtener una solución satisfactoria. La única posibilidad ofrecida fue modificar la geometría de la molécula para adaptarse a la base de datos de valores precalculados ofrecidos por PennyLane.
- Se obtuvieron resultados muy diferentes con distintas versiones de PennyLane (se probaron la 0.26 y la 0.27). Se reportó a PennyLane, que no ha solucionado el problema hasta la fecha.
- Para intentar acelerar el proceso, se sustituyó el uso del descenso del gradiente por el cálculo del quantum natural gradient, que debería ofrecer una ventaja en tiempo de ejecución. Se obtuvo justo el efecto contrario. También se sustituyó el simulador por defecto por el simulador *lightning*, que ofrece prestaciones mejoradas. De nuevo, la ejecución se ralentizó en lugar de acelerarse. Esto fue también reportado a Xanadu, sin encontrar respuesta hasta el momento.

En vistas de las múltiples dificultades, muy posiblemente causadas por errores de programación de la librería PennyLane, se decidió abandonar el uso de esta herramienta para las pruebas subsiguientes.

Determinación de la energía del ground state de una molécula de metano

Con el objetivo de comprobar si el método del VQE podía ser utilizado en simuladores cuánticos con moléculas más complejas que el agua, se seleccionó como siguiente caso de uso la molécula de metano (CH_4). La geometría seleccionada fue la siguiente:

```
mol = Molecule(geometry=[['c',  [0.000000,  0.000000,  0.000000]],
['H',  [0.000000,  0.000000,  1.089000]],
['H',  [1.089000,  0.000000,  0.000000]],
['H',  [-0.544500,  -0.943102,  0.000000]],
['H',  [-0.544500,  0.943102,  0.000000]]])
```

Pruebas con Qiskit Nature 0.4.5

Se utilizó la misma implementación que, con la librería Qiskit Nature en su versión 0.4.5, había sido usada para la determinación del estado de mínima energía de la molécula de agua. En primer lugar, se obtuvo el hamiltoniano fermiónico. La expresión de este hamiltoniano involucra 19894 términos, lo que supone un incremento de casi cuatro veces con respecto a la molécula de agua. Los primeros términos del hamiltoniano son los siguientes:

```
-19.687659089191495 * ( +_0 -_0 )
+ 0.3590231653878759 * ( +_0 -_1 )
+ -3.476380121043512e-08 * ( +_0 -_2 )
+ -0.09363923142715412 * ( +_0 -_4 )
```

```
+ 0.1985128473503342 * ( +_0 -_5 )
+ -9.478673208 ...
```

A continuación, usando la transformación de Jordan-Wigner, se obtuvo el hamiltoniano expresado con matrices de Pauli. Para este hamiltoniano, que presenta miles de términos, es necesario utilizar 18 qubits. Su expresión comienza con los siguientes valores:

```
-37.17967405794578 * IIIIIIIIIIIIIIII
+ 6.2627008623946905 * IIIIIIIIIIIIIIIZ
+ 0.06436414298940366 * IIIIIIIIIIIIIIYY
+ 0.06436414298940366 * IIIIIIIIIIIIIIXX
- 0.013098109629198612 * IIIIIIIIIIIYZZZY
- 0.013098109629198612 * IIIIIIIIIIIXZZZX
+ 0.031648131922968975 * IIIIIIIIIIYZZZZY
+ 0.031648131922968975 * IIIIIIIIIIXZZZZX
- 2.41410087433114e-07 * IIIIIIIIYZZZZZZY
- 2.41410087433114e-07 * IIIIIIIIXZZZZZZX
+ 0.05604120158999705 * IIIIIIIIYZZZZZZZY
+ 0.05604120158999705 * IIIIIIIIXZZZZZZZX
+ 0.6564684007983824 * IIIIIIIIIIIIIIZI
+ 1.1601182382431098e-08 * IIIIIIIIIIIIIIYYI
+ 1.1601182382431098e-08 * IIIIIIIIIIIIIIXXI
```

Cuando se intentó ejecutar el método VQE (con la forma variacional UCCSD), Qiskit arrojó un error causado por la superación del límite para el tamaño de las matrices que es posible utilizar con este algoritmo.

Pruebas con Qiskit Nature 0.5.0

En vista de los problemas experimentados con la versión 0.4.5 de Qiskit Nature en la ejecución del algoritmo VQE con la molécula de metano, se decidió realizar las pruebas con la versión 0.5.0 de la librería. Con esta actualización, es posible ejecutar VQE con un hamiltoniano de 18 qubits sin superar el límite de tamaño impuesto a las matrices.

Sin embargo, tras tres semanas de ejecución continuada del algoritmo no se había obtenido ningún resultado, por lo que se decidió interrumpir la ejecución. Para intentar averiguar la causa de la ralentización del proceso, se realizó una nueva implementación con callbacks para obtener información tras cada iteración del algoritmo de minimización. Como resultado se obtuvieron los siguientes datos:

- Cada iteración emplea, aproximadamente, unos 40 minutos.
- En cada iteración, al menos durante la exploración inicial, se modifica un único parámetro.
- El número de parámetros de la forma variacional es cercano a 200.

Por tanto, solo para la exploración inicial de los parámetros serían necesarias más de 120 horas de computación. Esto explica el hecho de no haber obtenido resultados tras tres semanas de cálculo y parece indicar que no resulta recomendable ejecutar el algoritmo VQE con moléculas medianamente complejas en simuladores clásicos de circuitos cuánticos.

PT6: Validación de la viabilidad de las soluciones cuánticas

Este paquete de trabajo se organiza en torno a dos tareas, ambas iniciadas y terminadas en la anualidad 2023.

- T6.1: Validación de resultados (iniciada y terminada en 2023).
- T6.2: Ajustes de modelos (iniciada y terminada en 2023).

T6.1: Validación de resultados

A continuación, se muestran las actividades llevadas a cabo dentro del marco de esta tarea:

Clasificación

- Una vez presentados los resultados, se procede a analizar la viabilidad de las versiones cuánticas para problemas de clasificación.
- Tomaremos como referencia las tablas presentadas y juntando los resultados del mejor modelo clásico de clasificación con su contraparte cuántica en una sola tabla obtenemos lo siguiente:

	Modelo	Accuracy	precision	Recall	F1	Tiempo entrenamiento medio	Tiempo prediccion medio
0	Variational	0.75	0.71	0.83	0.79	32.34	3.89
1	Decision Tree	0.98	0.99	0.98	0.99	1.19	0.04

-
- Como se mencionó anteriormente, los modelos clásicos de clasificación han obtenido mejores resultados de rendimiento de las predicciones en todas las métricas estudiadas mostrando que son capaces de distinguir las trazas que suponen ciberataques de las que no con mayor claridad que los modelos cuánticos. Además, los modelos clásicos son capaces de conseguir esto en tiempos de ejecución considerablemente menores en el entrenamiento y algo menores en los de inferencia. Esto denota que la computación cuántica aún no se encuentra en el punto donde las limitaciones técnicas no permiten obtener ventajas claras sobre los modelos clásicos. Sin embargo hay algunas observaciones obtenidas a lo largo del proyecto que nos indican que una vez superadas estas limitaciones, los modelos cuánticos poseen cierto potencial con respecto a los modelos clásicos:
- Menos número de variables para la inferencia: los modelos cuánticos se han entrenado empleando únicamente 2 variables lo que denota que, a pesar de que los resultados obtenidos son peores que los clásicos, han requerido considerablemente menos información para lograr un rendimiento bastante aceptable, por encima del 70%.
- Menor cantidad de instancias en el conjunto de entrenamiento: de igual manera que ocurre con el punto anterior, los modelos cuánticos se han entrenado sobre un subconjunto de los datos significamente más pequeño que el conjunto original denotando que es capaz de aprender las características de los datos con un conjunto de entrenamiento considerablemente menor lo que para ciertos problemas de machine learning puede ser beneficioso ya que la falta de acceso a grandes conjuntos de datos para resolver problemas concretos es un problema a la orden del día, además de que supondría un requerimiento de memoria inferior con respecto a los modelos clásicos.
- Los tiempos de inferencia no son tan dispares: una de las características de los modelos variacionales es que, una vez que sus parámetros han sido entrenados, las inferencias de nuevos datos son bastante rápidas ya que sólo han de pasar por el circuito una vez agilizando

el proceso de predicción y haciendo que una vez que el modelo se ha entrenado, su aplicación práctica no sea muy distinta de modelos clásicos en términos de tiempos de predicción.

- Posible asociación entre la calidad de las variables y la correlación de Kendall: a lo largo de las pruebas realizadas en el proyecto se ha observado el rendimiento de los modelos cuánticos al utilizar diferentes subconjuntos de variables. Aquellas con mayor correlación de Kendall parecen ser las más efectivas para la clasificación binaria y podría servir como indicativo de la selección de variables para modelos de clasificación cuánticos a diferencia de los modelos clásicos donde no existe un consenso claro sobre la selección óptima de las variables del modelo.

Optimización

Aunque los algoritmos cuánticos ofrecen ventajas teóricas en ciertos problemas, la complejidad de traducir eficientemente algoritmos clásicos de optimización de rutas a su equivalente cuántico es un desafío considerable. Además, la naturaleza sensible de los cálculos cuánticos a perturbaciones externas y errores presenta un obstáculo significativo. La corrección de errores en este campo aún no ha alcanzado un nivel que permita manejar de manera efectiva problemas complejos de optimización como el Problema del Viajante (TSP).
Por otro lado, la infraestructura necesaria para implementar algoritmos cuánticos de optimización de rutas en empresas de transporte y logística no está disponible de manera generalizada. La falta de accesibilidad a sistemas cuánticos de calidad y su complejidad de integración en entornos empresariales limita su adopción práctica.
Si nos centramos en los resultados obtenidos en el presente trabajo, vemos que el coste temporal de la solución cuántica sobrepasa por mucho a la del algoritmo de computación clásica. Además de eso, se puede garantizar que la solución que arroja el algoritmo convencional es óptima, frente al de su contrario cuántico, con el que no podemos estar seguros de ello.

Coste del algoritmo tradicional	**142.054 ms**
Coste del algoritmo cuántico	**12.629.219 ms**

A día de hoy, la aplicabilidad de la computación cuántica en la optimización de rutas para empresas de transporte y logística se ve limitada por desafíos algorítmicos, errores cuánticos, requisitos de coherencia, y limitaciones de infraestructura y accesibilidad. La computación cuántica no ofrece ventajas prácticas sustanciales sobre la computación clásica en este contexto específico.

Simulación

Esta sección discute las conclusiones principales de los problemas de simulación química. El resultado más obvio es que los algoritmos y herramientas específicas para la simulación química están en una fase más inestable y preliminar comparados con sus contrapartes para problemas de clasificación y optimización, aunque estos últimos también están en una etapa temprana.

La obtención de resultados para las moléculas de dihidrógeno, agua y metano a través de métodos clásicos no presenta ningún reto técnico, ya que los resultados teóricos son fácilmente accesibles. Sin embargo, el uso de estas moléculas, tan simples, es un requerimiento básico para poder ejecutar el proceso de simulación basado en computación cuántica. Moléculas más complejas, que representan problemas reales de interés, resultan ser totalmente inviables—los costes de memoria y CPU explotan hasta resultar imposibles.

Se ha implementado el algoritmo VQE (Variational Quantum Eigensolver) en dos librerías diferentes: PennyLane y Qiskit, utilizando varias versiones de cada una. Los experimentos se realizaron con dos moléculas, agua y metano, empleando dos configuraciones variacionales distintas.

Para la validación del algoritmo VQE, se realizaron pruebas con la molécula de dihidrógeno. En estos casos, los valores de energía obtenidos con Qiskit y PennyLane coincidieron con los calculados mediante métodos clásicos. Con Qiskit, se logró obtener un perfil de disociación completo para esta molécula.

No obstante, simular moléculas más complejas presentó significativas dificultades al usar simuladores cuánticos. Para la molécula de agua, se lograron reproducir los resultados de los algoritmos clásicos únicamente con Qiskit y después de un tiempo de ejecución notablemente largo. La simulación de la molécula de metano no pudo completarse. Se identificaron numerosos problemas con el software actual, especialmente con PennyLane.

La conclusión de estos experimentos indica que **el método VQE debe ejecutarse con ordenadores cuánticos reales, no simuladores, para lograr ventajas sobre los algoritmos clásicos**. Es crucial seleccionar adecuadamente la forma variacional y aplicar técnicas para reducir el tamaño de los hamiltonianos utilizados, explotando simetrías del sistema cuando sea posible. Además, **es esencial contar con librerías de programación cuántica más maduras y con funcionalidades específicas para la simulación molecular**.

Hay que ver estos problemas en perspectiva, considerando que la limitada disponibilidad de ordenadores cuánticos nos ha llevado a utilizar un simulador cuántico en un ordenador clásico. Esto implica limitaciones en potencia y en los requisitos de sistema necesarios. El software para cálculos DFT en ordenadores cuánticos no está tan avanzado ni establecido como el de los ordenadores clásicos, resultando en menos información y soporte disponibles para abordar problemas que surgen durante el proyecto. De hecho, investigaciones previas en este campo tampoco han trabajado con moléculas más grandes que las utilizadas aquí.

T6.2: Ajustes de modelos

Puesto que los modelos de clasificación son los únicos que han permitido cierta mejora mediante el ajuste de sus parámetros, las actividades llevadas a cabo dentro de esta tarea y también descritas en el E6.2 “Ajustes de modelos” se centran exclusivamente en estos modelos.

Diseño e implementación de los modelos

En esta sección se muestra el análisis realizado sobre las diferentes posibles implementaciones para poder comparar cada una en su versión óptima para el problema en cuestión.

Quantum Variational Classifier

Dado que uno de los componentes más importantes (si no el que más) de los clasificadores variacionales cuánticos es el optimizador (Cerezo, y otros, 2021), se ha llevado a cabo un análisis para determinar cuál de ellos es el mejor para este modelo y problema, así como la tasa de aprendizaje.

Se han llevado a cabo dos pruebas diferentes. La primera para determinar el mejor optimizador con los parámetros de los mismos por defecto y la segunda alterando las tasas de aprendizaje para ver si estas tienen un impacto en el rendimiento y el resultado final. Para acotar la variabilidad del modelo se ha decidido lanzar cada prueba con 1.000 repeticiones para cada optimizador en la primera prueba, y para cada optimizador y tasa de aprendizaje en la segunda.

Circuito cuántico

El circuito cuántico utilizado para el clasificador variacional cuántico sigue una estructura similar a (Schuld, Bocharov, Svore, & Wiebe, 2018) y, a nivel de código, utiliza las plantillas proporcionadas por el entorno Pennylane (PennyLane Variational classifier, 2021) (PennyLane AngleEmbedding, s.f.) (PennyLane StronglyEntanglingLayers, s.f.). Este circuito consta de dos bloques principales:

- AngleEmbedding: Este bloque se encarga de asignar los valores reales a ángulos de cada qubit.
- StrongEntanglingLayers: Este bloque trata de entrelazar fuertemente todos los qubits del circuito de manera que al alterar el valor de uno de ellos, alterará a todos los demás. Tal y como se sugiere en (Schuld, Bocharov, Svore, & Wiebe, 2018), esto es necesario para que el circuito alcance una posible solución.

Juntando ambos bloques, el circuito queda entonces de la siguiente forma:

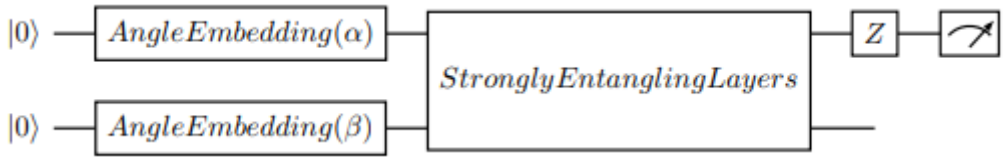


Figura 3. Circuito variacional general.

El bloque AngleEmbedding puede emplearse para rotar el qubit en cualquiera de los 3 ejes. En este caso se ha realizado sobre el eje X y por tanto se puede representar de la siguiente manera:

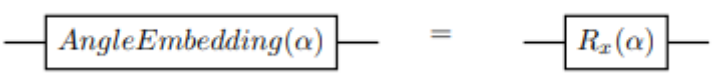


Figura 4. Equivalencia del circuito AngleEmbedding.

Por otra parte, el bloque StrongEntanglingLayers se representa como:

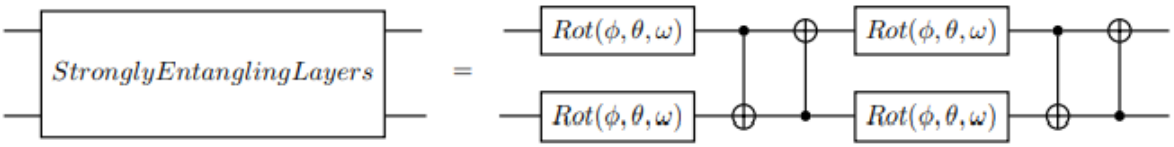


Figura 5. Equivalencia del bloque StrongEntanglingLayers de Pennylane mediante puertas Rot.

Donde el bloque $Rot(\phi, \theta, \omega)$ se representa como las puerta unitarias:

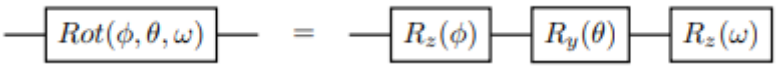


Figura 6. Equivalencia de la puerta Rot de Pennylane.

Quedando StrongEntanglingLayers como:

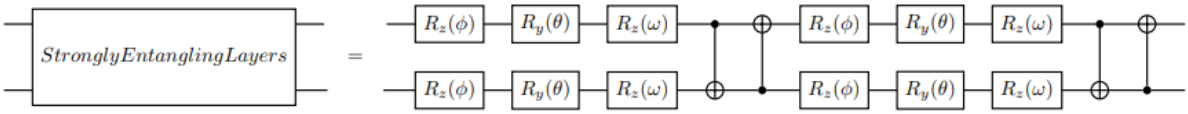


Figura 7. Equivalencia completa del bloque StrongEntanglingLayers de Pennylane.

Sustituyendo los bloques por sus equivalencias, el circuito final sería el siguiente:

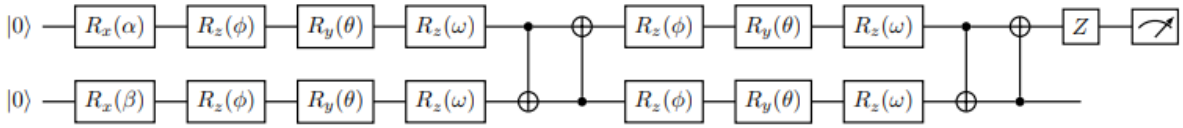


Figura 8. Circuito variacional cuántico final.

Parámetros básicos

Además del optimizador, existen otros parámetros que necesita el modelo para funcionar correctamente. Tras algunas pruebas básicas se ha determinado que, para este problema, la configuración que proporciona la mejor relación entre precisión y tiempo de ejecución es la siguiente:

- Número de qubits del circuito cuántico: 2
- Número de capas del circuito: 2
- Tamaño de bloque de training: 2
- Número de iteraciones: 50
- Precisión mínima para converger: 95%

Optimizador

A continuación, se presentan los resultados obtenidos en la prueba de optimizadores en términos de precisión, tiempo de entrenamiento y tiempo de predicción en formato de histograma, diagrama de violín y diagrama de caja.

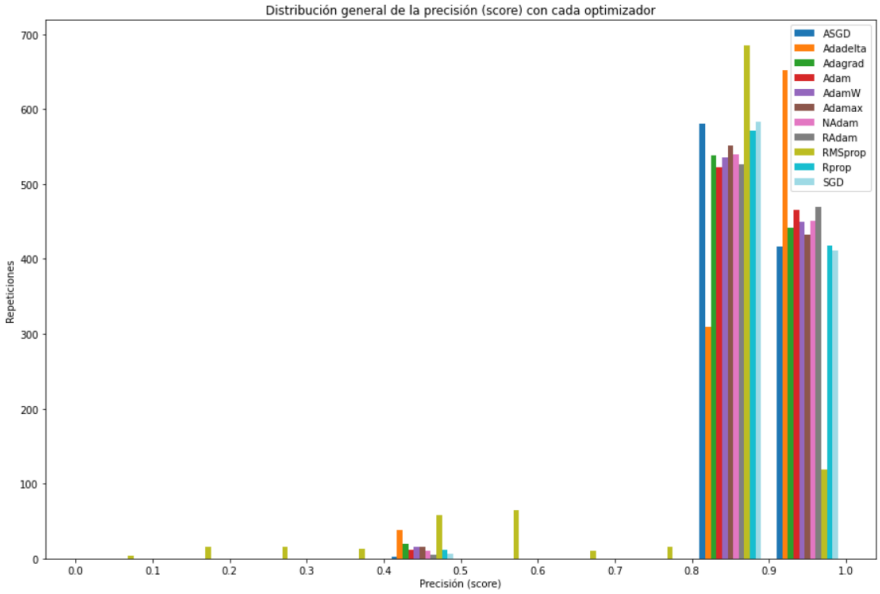


Figura 9. Histograma de precisión con cada optimizador.

En términos de la precisión del modelo, no se aprecian grandes diferencias al elegir diferentes optimizadores, con la excepción del optimizador RMSprop, el cual añade más aleatoriedad al modelo y por tanto el intervalo de posibles valores de la precisión del modelo es mucho mayor.

Por otra parte, se aprecia que, en la gran mayoría de los casos, el modelo obtiene una precisión de entre el 80 y el 100% y en un reducido número de casos, el modelo se estanca entre el 40 y el 50%. Estos últimos son los casos en los que el modelo no ha podido converger a la solución debido al subconjunto de valores de entrenamiento elegidos durante el mismo.

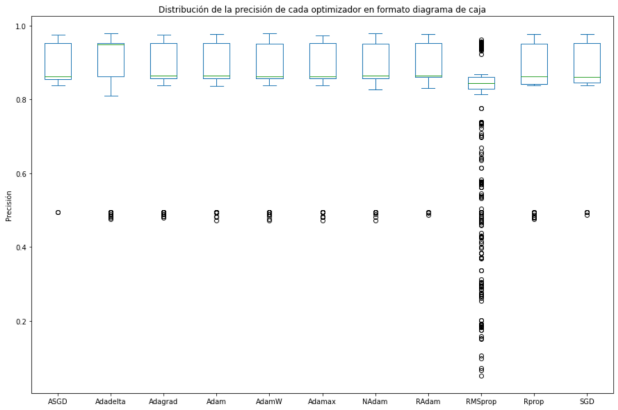


Figura 10. Diagrama de caja de la precisión con cada optimizador.

La representación en diagrama de caja muestra que al menos el 75% de los datos se encuentran en la franja de entre el 80 y el 100% de precisión, destacando el caso de Adadelta donde la mediana es superior a las demás y el caso del RMSprop el cual contiene múltiples valores extremos o extraños debido a su naturaleza aleatoria.

En cuanto a los tiempos de entrenamiento se ha decidido utilizar un histograma apilado.

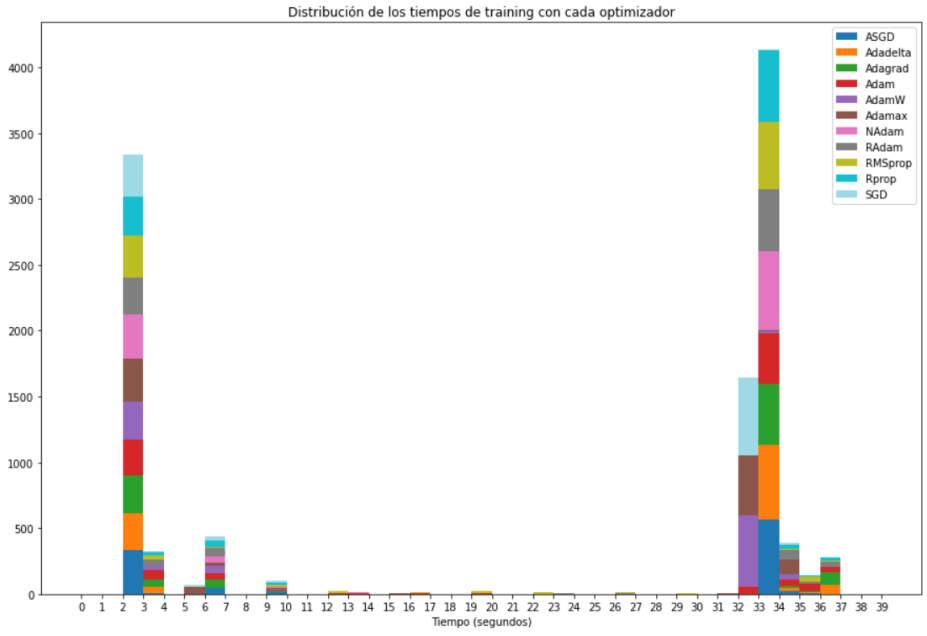


Figura 11. Histograma acumulado de los tiempos de entrenamiento con cada optimizador.

Se puede apreciar los casos en los que el modelo converge rápidamente y aquellos en los que no, dependiendo de la selección inicial de los parámetros y del subconjunto de datos de entrenamiento. De nuevo, no se aprecian grandes diferencias entre los diferentes optimizadores en este aspecto.

Para comprobar la presencia de cada optimizador en cada uno de los extremos del gráfico, se han construido otros dos gráficos:

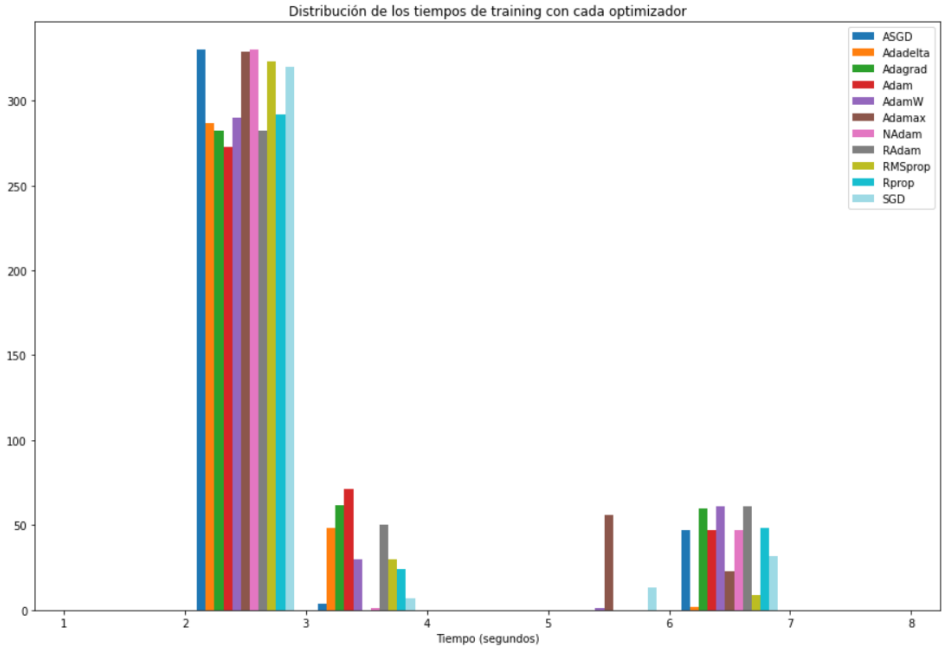


Figura 12. Histograma de la sección donde el algoritmo converge en las primeras iteraciones con cada optimizador.

Aquí se muestra la presencia de cada optimizador en las secciones donde el modelo converge rápidamente y encuentra una solución cercana al 95% de precisión. En este caso no se aprecian grandes diferencias entre los diferentes optimizadores que indiquen que un optimizador tiende a llegar a esta solución con una probabilidad mayor.

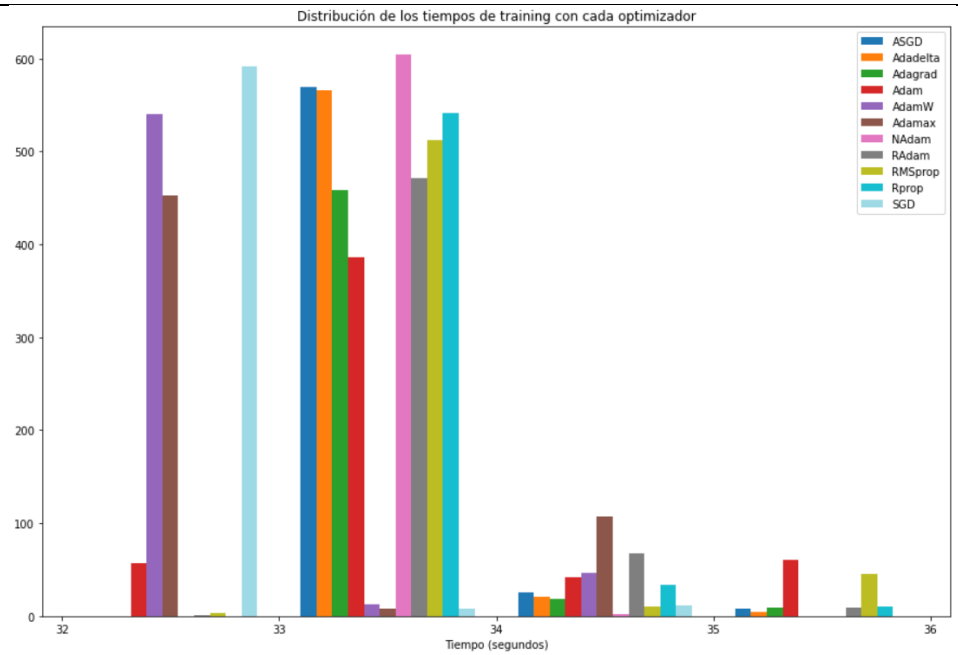


Figura 13. Histograma de la sección donde el modelo no converge al 95% con cada optimizador.

En cuanto al otro extremo de la gráfica, se puede destacar el optimizador AdamW como el que, dentro de este intervalo, posee mayor presencia en los tiempos más bajos, indicando que es más probable que complete el entrenamiento más rápido.

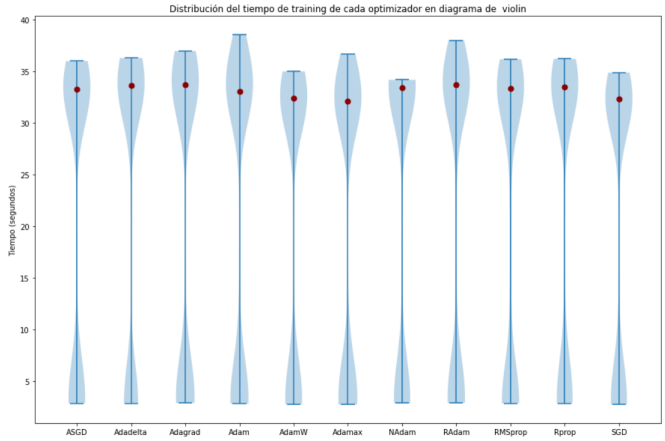


Figura 14. Diagrama de violín de los tiempos de entrenamiento con cada optimizador.

Para representar los tiempos de entrenamiento, se ha utilizado únicamente el diagrama de violín y no el de caja ya que este último puede llevar a una idea equivocada sobre la distribución de los datos en conjuntos de datos como este donde la mayor parte de estos se concentran en los extremos del intervalo. Aquí se puede ver que la mediana se encuentra entre los 30 y 35 segundos, por lo que al menos la mitad de las veces el modelo no convergerá en las primeras iteraciones, sino que requerirá más o incluso no convergerá. Esto no indica sin embargo que el modelo no sea preciso ya que, como

se ha mencionado anteriormente, se ha seleccionado una precisión del 95% como requisito para converger, por lo que puede obtener un nivel alto de precisión, pero no llegar a este punto.

Para elegir el mejor optimizador en este aspecto, se ha combinado el valor medio de precisión y tiempo de entrenamiento del modelo con los diferentes optimizadores y se ha representado en la siguiente gráfica:

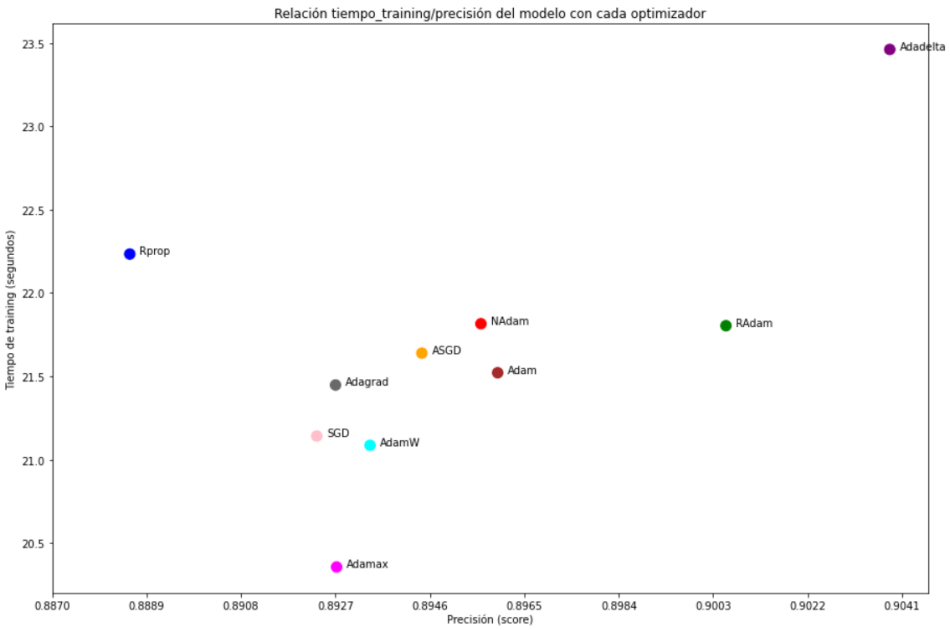


Figura 15. Relación de tiempo de entrenamiento y precisión con cada optimizador.

En esta se puede ver que existen dos optimizadores que destacan. Por una parte, el optimizador Adamax es el más rápido, pero con una de las peores precisiones de media y, por otra parte, el optimizador Adadelta, que posee la mayor precisión, pero el peor tiempo de entrenamiento de media. Dado que las diferencias absolutas entre las precisiones de todos los optimizadores son casi despreciables, se puede considerar que a priori el optimizador Adamax es la mejor opción.

En cuanto a los tiempos de predicción se han representado mediante un histograma sobre diferentes intervalos de tiempo:

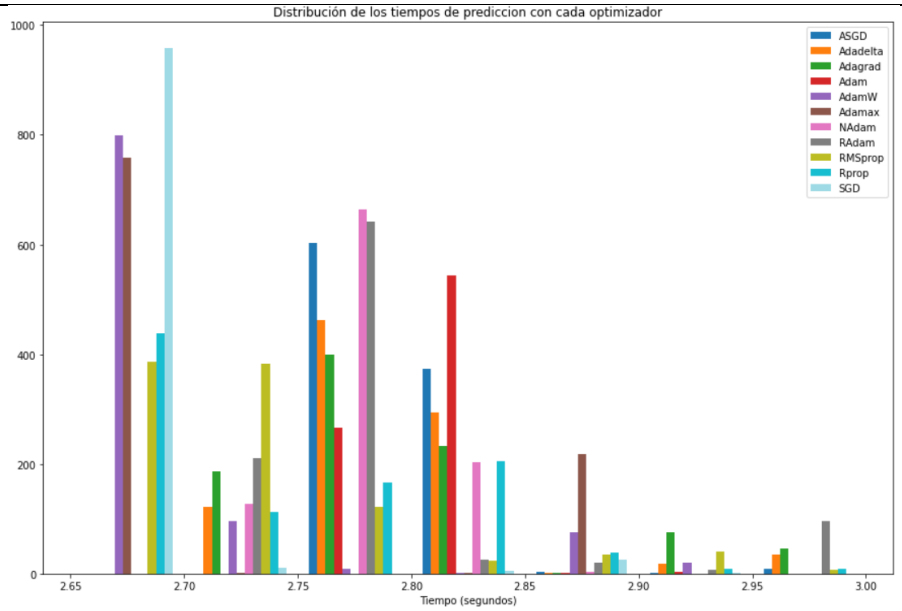


Figura 16. Histograma de los tiempos de predicción con cada optimizador.

Aquí las diferencias existentes entre los diferentes optimizadores son despreciables ya que la mayor diferencia entre ambos extremos no supera siquiera las 5 décimas de segundo por lo que en este aspecto se considera que todos los optimizadores tienen el mismo rendimiento.

Optimizador y tasa de aprendizaje

A continuación, se presentan los resultados de la segunda prueba, en la que se ha lanzado el modelo con diferentes optimizadores y diferentes tasas de aprendizaje para comprobar el impacto de estas en el modelo final.

Para representar la precisión, el tiempo de entrenamiento y el tiempo de predicción se han utilizado diagramas de barras donde se muestra el valor medio de cada una de estas características para cada modelo y tasa de aprendizaje.

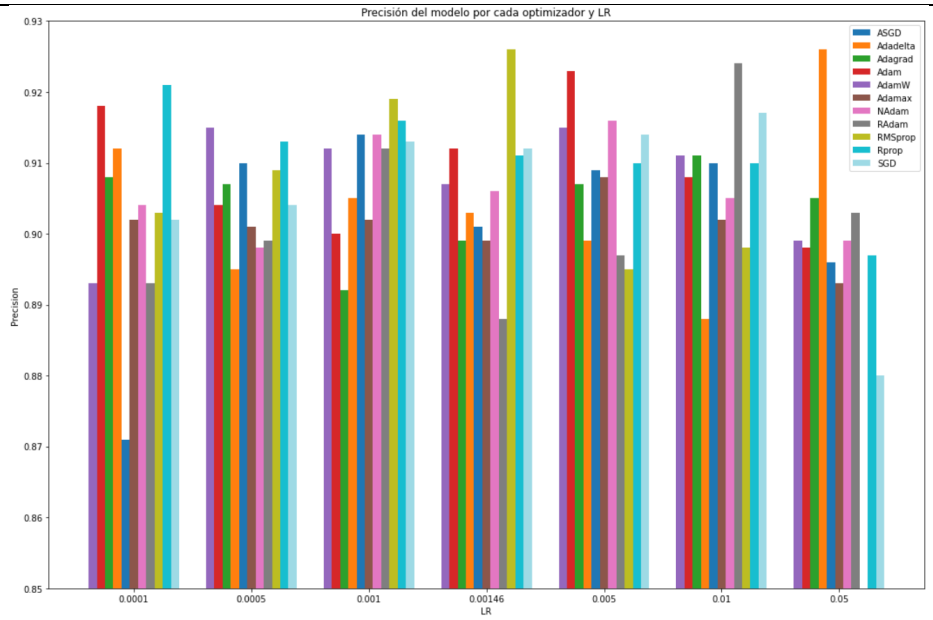


Figura 17. Diagrama de barras de la precisión media con cada optimizador y tasa de aprendizaje.

En cuanto a la precisión se pueden destacar lo siguiente:

- RMSprop y tasa de aprendizaje 0.00146
- Adam y tasa de aprendizaje 0.005
- Radam y tasa de aprendizaje 0.01
- Adadelta y tasa de aprendizaje 0.05

Todos ellos con precisiones entre el 92 y 93% de media a priori son los mejores candidatos.

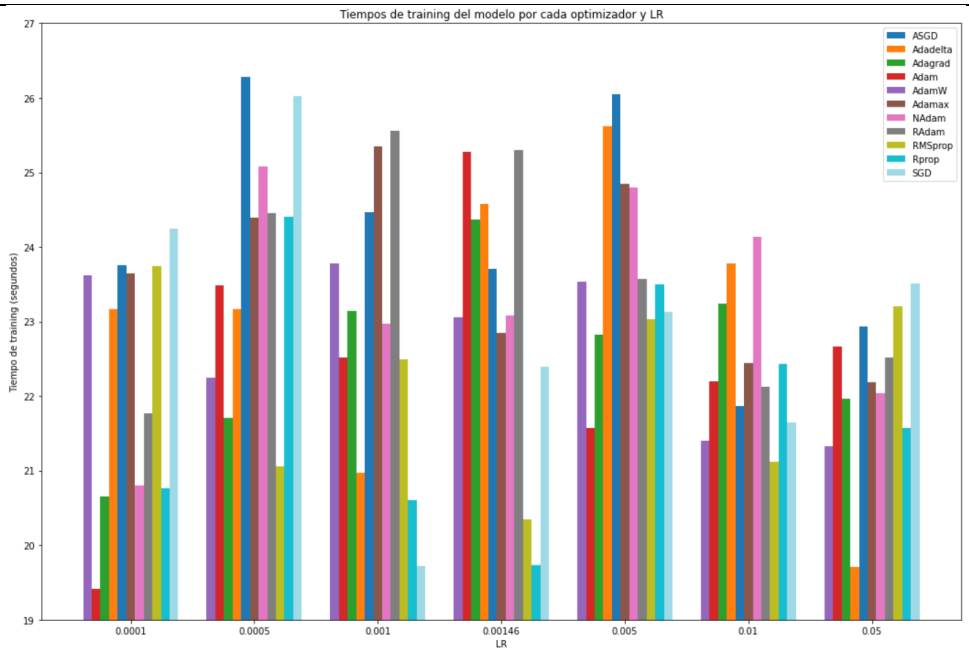


Figura 18. Diagrama de barras del tiempo de entrenamiento medio con cada optimizador y tasa de aprendizaje.

Los tiempos de entrenamiento son variables, destacando en este caso:

- Adam y tasa de aprendizaje 0.0001
- SGD y tasa de aprendizaje 0.001
- Rprop y tasa de aprendizaje 0.00146
- Adadelta y tasa de aprendizaje 0.05

Al igual que ocurría en la prueba anterior, la diferencia en cuanto a los tiempos de predicción entre todos los candidatos es despreciable como se puede ver en la siguiente gráfica:

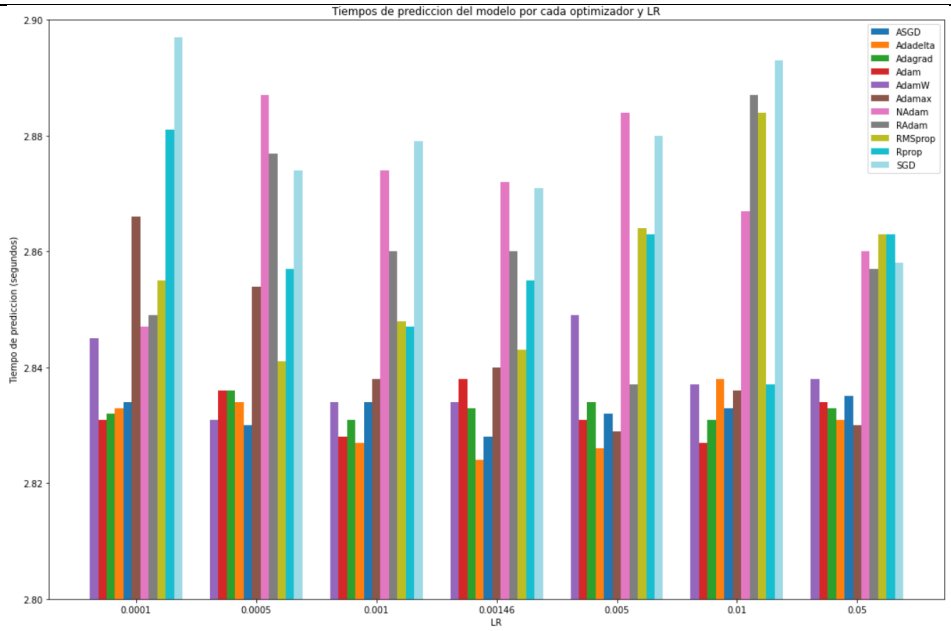


Figura 19. Diagrama de barras del tiempo de predicción medio con cada optimizador y tasa de aprendizaje.

Por lo que de nuevo se considera que en este aspecto su rendimiento es idéntico.

Combinando la precisión y el tiempo de entrenamiento medios de cada optimizador y tasa de aprendizaje se obtiene la siguiente gráfica de nube de puntos:

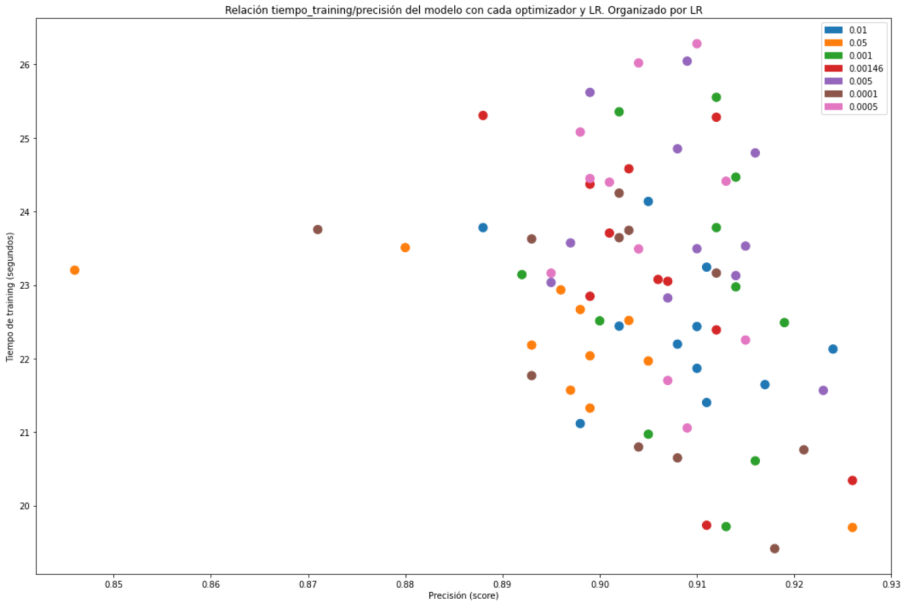


Figura 20. Nube de puntos representando la relación entre tiempo de entrenamiento y precisión medios de cada modelo y clasificado por tasas de aprendizaje.

Estos se han clasificado por tasas de aprendizaje y se puede ver que de entre todos ellos destaca un punto perteneciente a la tasa con valor 0.05 el cual posee la mejor relación entre tiempo de

entrenamiento y precisión medios. Por ello se han filtrado los optimizadores con esta tasa de aprendizaje dando como resultado:

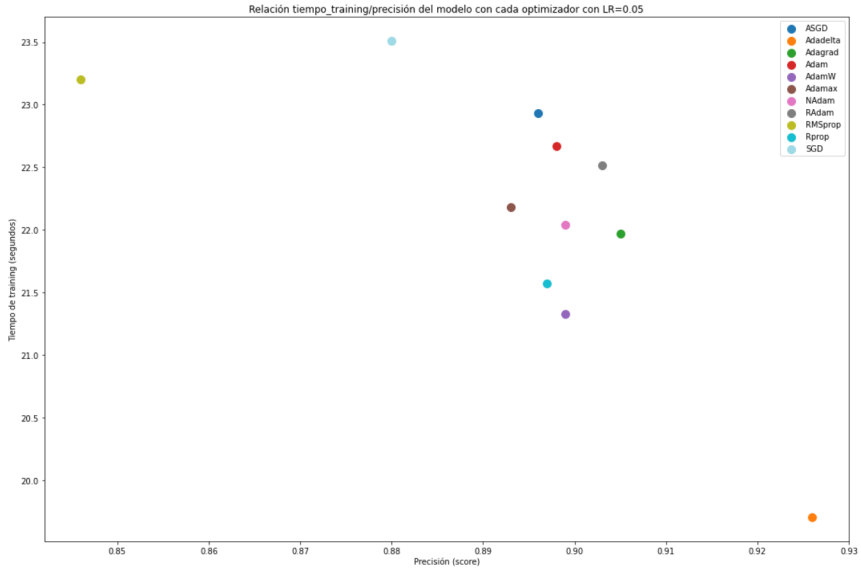


Figura 21. Relación entre tiempo de entrenamiento y precisión medias de cada modelo con tasa de aprendizaje 0.05.

Donde se ha detectado que dicho punto pertenece al optimizador Adadelta, por lo que finalmente la mejor configuración de este modelo para este problema en concreto es el optimizador Adadelta con tasa de aprendizaje 0.05.

Kernel

El modelo de clasificación cuántica basado en Kernels está basado en las versiones implementadas en (PennyLane, 2021) y (PennyLane Training and evaluating quantum kernels, 2021). Dado que este utiliza en gran medida la librería Scikit learn para su implementación, el único aspecto que puede modificarse para mejorar el rendimiento de este modelo son los datos de entrada.

Se han realizado diferentes tipos de preprocesado de los mismos para comprobar el impacto que pueden tener sobre su rendimiento, pero al no encontrar diferencias sustanciales se han mantenido tal cual están.

Circuito cuántico

El único aspecto que se puede comentar del modelo es el circuito cuántico que utiliza y es el siguiente:

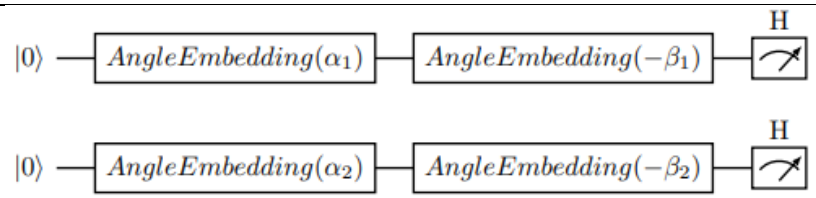


Figura 22. Circuito cuántico general del modelo basado en Kernels.

El cual, utilizando las equivalencias mostradas en el circuito variacional cuántico, puede representarse con puertas unitarias de la siguiente manera:

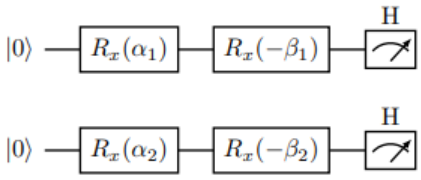


Figura 23. Circuito cuántico real del modelo basado en kernels.

Aquí hay que tener en cuenta que se utiliza una notación diferente para la evaluación del circuito, ya que esta se realiza respecto del estado inicial |0..0⟩⟨0..0| y para ellos se utiliza la matriz Hermitiana.

Quantum K-Nearest Neighbours

El modelo cuántico de K-vecinos empleado está basado en el implementado en (Ket.G, 2022).

Al igual que el caso del modelo basado en Kernels, el modelo cuántico de K-vecinos utiliza en gran medida la librería Scikit learn para entrenar el modelo, utilizando únicamente el circuito para calcular la distancia entre vecinos. Sin embargo, en este caso es posible paralelizar parte del proceso, más concretamente se ha decidido paralelizar la parte de predicción del modelo ya que esta requiere considerablemente más tiempo que el entrenamiento.

Circuito cuántico

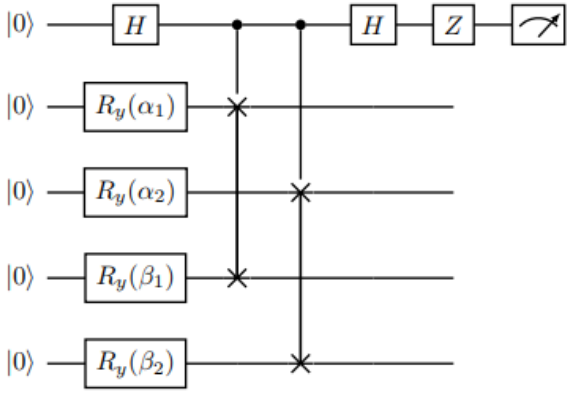


Figura 24. Circuito cuántico del modelo K-Vecinos.

Número óptimo de unidades de procesamiento

Dado que se ha paralelizado el modelo cuántico de K-vecinos, se ha optado por realizar un pequeño análisis para determinar el número óptimo de unidades de procesamiento. Se ha lanzado el modelo en serie y paralelizado con un número par de cores escalando desde 2 hasta 64 cores obteniendo la siguiente gráfica de tiempos de predicción:

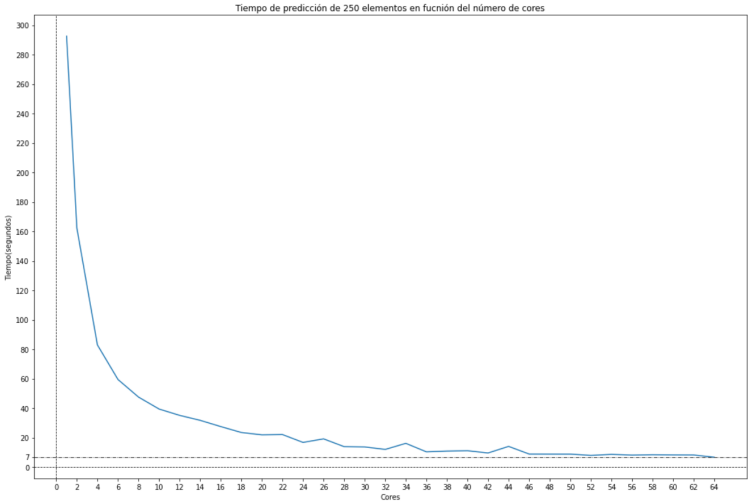


Figura 25. Evolución del tiempo de predicción del modelo K-vecinos con el número de unidades de procesamiento.

Se puede observar cómo el tiempo de ejecución disminuye a medida que aumentamos el número de cores hasta llegar a un punto en el que dicho tiempo de ejecución apenas disminuye con el número de cores y se estanca alrededor de los 7 segundos.

Con estos datos se ha calculado el Speedup para comprobar cuán rápido es el algoritmo a medida que aumentamos el número de cores con respecto a la versión en serie. El Speedup se calcula con la siguiente fórmula:

$$Speedup = \frac{T_{Serie}}{T_{Paralelo}}$$

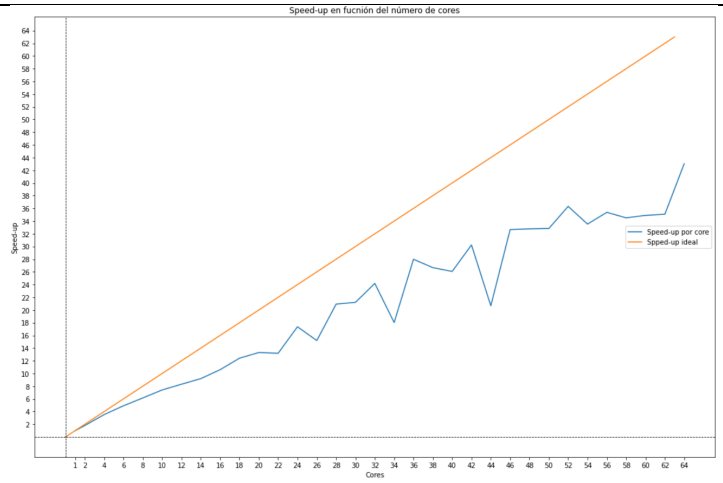


Figura 26. Speedup y Speedup ideal del modelo K-vecinos paralelizado.

En la anterior gráfica se puede ver que, en un contexto ideal, el Speedup debería seguir una tendencia lineal siendo X veces más rápido que la versión en serie, siendo X el número de cores utilizados. Sin embargo, debido a la lógica extra requerida para coordinar cada uno de los procesos y por la propia ley de Amdahl, se puede ver que la realidad muestra que el Speedup no solo no sigue esta tendencia, sino que además se aleja más de ella con el número de cores. Además, en la gráfica se ve que existen picos en la recta del Speedup real. Esto es debido a que el problema a tratar debe ser dividido en un número de partes igual al número de cores utilizados para que cada uno de estos maneje la misma carga de trabajo, lo cual no siempre es posible, provocando que algunos cores realicen más tareas mientras que otros permanecen ociosos y desaprovechados.

Una vez calculado y analizado el Speedup del algoritmo, se ha procedido a calcular la evolución de la eficiencia con el número de cores. Esta se calcula mediante la siguiente fórmula:

$$Eficiencia = \frac{Speeup}{cores}$$

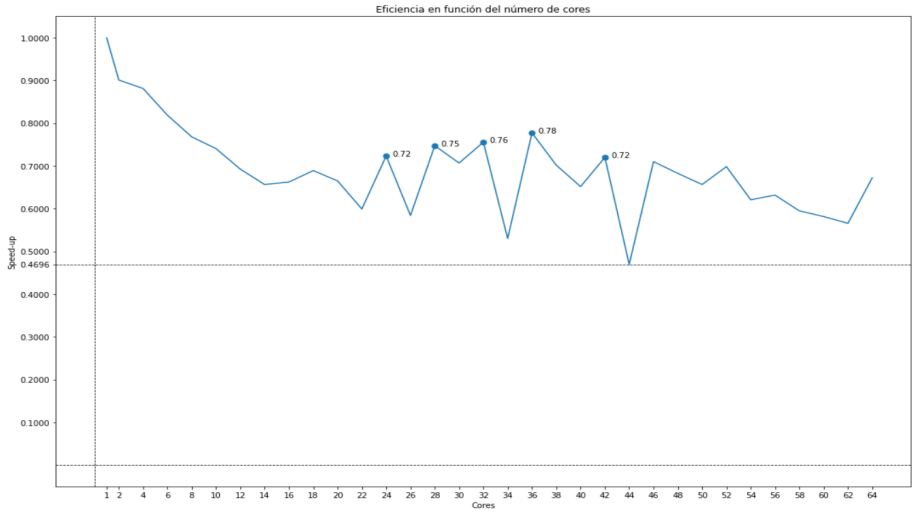


Figura 27. Eficiencia del modelo K-vecinos paralelizado.

La gráfica de la eficiencia escalada muestra que esta disminuye a medida que aumentamos el número de cores, tal y como sucedió con el Speedup, ya que ambos están relacionados. Además, aparecen de nuevo los picos en la gráfica por los motivos explicados anteriormente.

Para encontrar el número óptimo de unidades de procesamiento, se han calculado los máximos locales de la anterior gráfica en el intervalo de puntos a partir de los 12 cores, que es cuando esta empieza a aplanarse en la gráfica de tiempo de ejecución obteniendo los puntos mostrados en ella con sus respectivos valores de eficiencia. A priori se puede ver que la configuración con 36 cores resulta la más eficiente para el modelo.

Es importante explicar que, aunque existen puntos de la gráfica donde la eficiencia es mucho mayor, rozando incluso el 100% en el caso con 2 cores, esta solo indica el tiempo relativo obtenido con cada configuración de cores, por lo que se debe tener en cuenta también el tiempo absoluto del algoritmo. Es por ello que se ha construido la siguiente gráfica donde se muestra la relación entre eficiencia y tiempo de ejecución de cada configuración de cores:

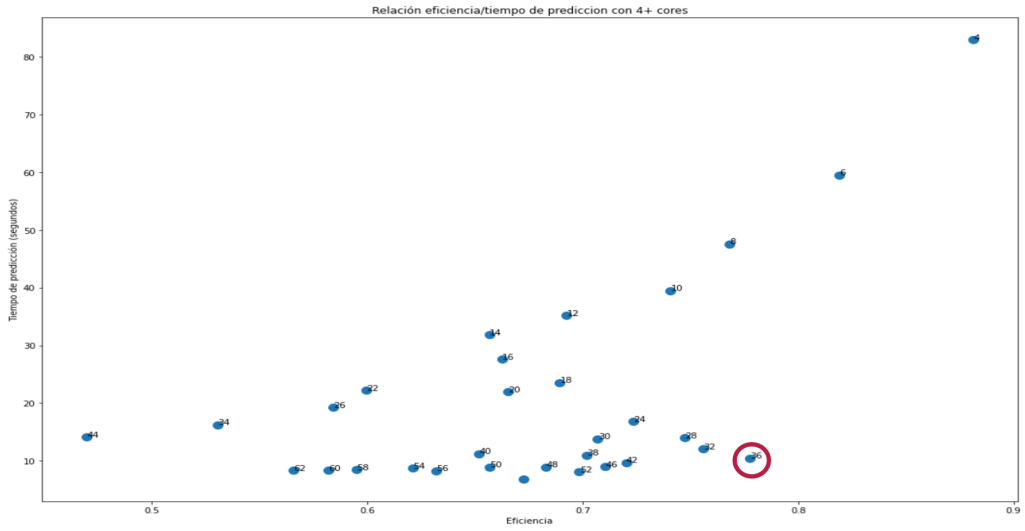


Figura 28. Relación entre tiempo de ejecución y eficiencia para cada configuración del modelo K-vecinos paralelizado.

Aquí se puede observar que la configuración con 36 cores es la que posee el mejor balance entre tiempo total de predicción y eficiencia con un tiempo medio de aproximadamente 10 segundos y una eficiencia cercana al 80%.

Resultados comparativos

Una vez definidos los modelos en su estado óptimo, se han realizado una serie de pruebas escaladas para comprobar su rendimiento y determinar el mejor de ellos. Para ello, se ha lanzado el modelo con diferentes tamaños del problema de manera incremental de 10 en 10 y se han medido los tiempos de entrenamiento y de predicción del modelo. Debido al rendimiento preliminar de alguno de los modelos y a la cantidad de tiempo que requeriría la ejecución con el tamaño del problema descrito

en la sección Análisis y Comparación del Conjunto de datos, se ha reducido el mismo a un máximo de 1000 elementos. Además, no se ha comparado la precisión en este caso, ya que los tres mantienen una precisión muy similar (alcanzando el 90%) y esta apenas tiene variaciones a medida que se aumenta el número de datos a tratar.

En primer lugar, se ha medido el rendimiento en cuanto a los tiempos de entrenamiento de los tres modelos, obteniendo la siguiente gráfica:

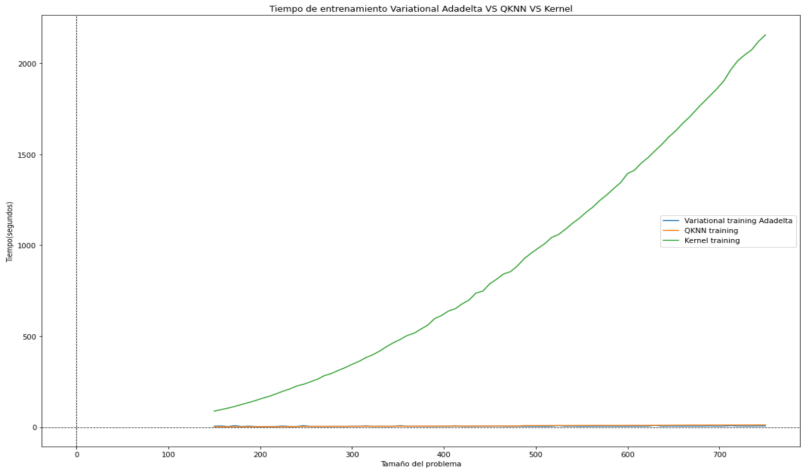


Figura 29. Gráfica comparativa del tiempo de entrenamiento escalado de los modelos Kernel, QKNN y QVC.

Como se puede observar, el modelo basado en Kernels obtiene el peor rendimiento en cuanto al tiempo de entrenamiento con diferencia, ya que este crece considerablemente más rápido que los otros dos modelos a medida que aumenta el tamaño del problema.

Como nota adicional, si se elimina el modelo K-vecinos de la gráfica anterior, se puede ver que la gráfica anterior coincide con la presentada en Pennylane, donde se representa el número de evaluaciones requeridas por el modelo basado en Kernels y el modelo variacional cuántico.

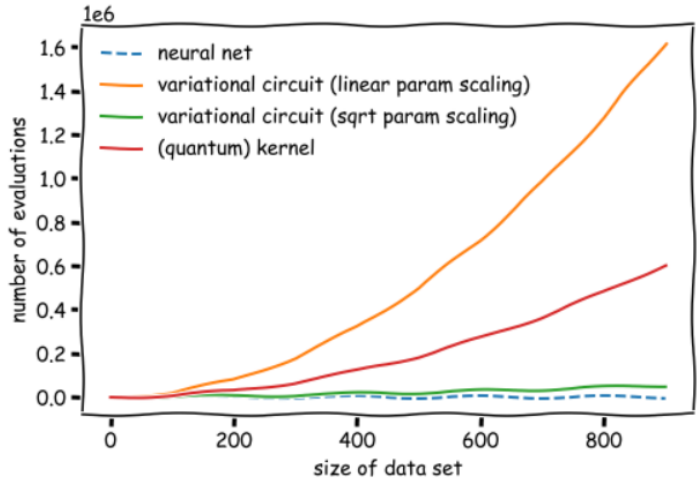


Figura 30. Gráfica comparativa del número de evaluaciones del circuito de los modelos Kernel, Variational con número de parámetros con relación lineal, Variational con número de parámetros con relación raíz cuadrada y una red neuronal obtenida de (PennyLane, 2021).

En esta se aprecia que el modelo Kernel crece a un ritmo mucho mayor siempre y cuando el número de parámetros a analizar por el modelo variacional cuántico no crezca linealmente con el tamaño del problema, que es el caso del modelo propuesto.

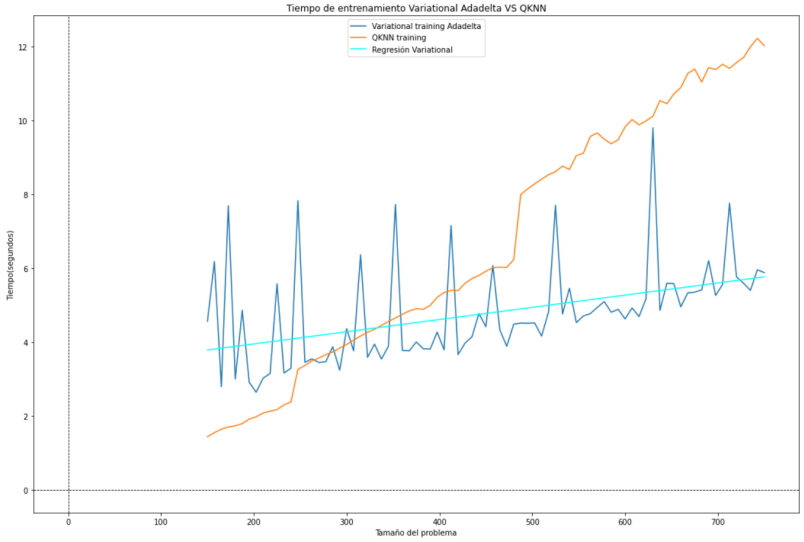


Figura 31. Gráfica comparativa del tiempo de entrenamiento escalado de los modelos QKNN y QVC además de una recta de regresión que aproxima QVC.

Si se elimina la versión basada en Kernels de la gráfica, se puede ver con mayor claridad las diferencias entre el modelo KNN cuántico y el modelo variacional cuántico. Lo primero que se puede observar son las diferencias entre tiempos que posee el modelo variacional que a priori no sigue una tendencia clara con respecto al tamaño del problema. Esto es debido a la naturaleza aleatoria mencionada anteriormente y para ello, se ha construido una recta de regresión que lo aproxime lo

mejor posible. Gracias a esta, se puede ver mejor la tendencia que siguen ambos modelos, donde se aprecia que el modelo variacional cuántico escala ligeramente mejor que el modelo cuántico de K-Vecinos.

En cuanto a los tiempos de predicción, se ha realizado un análisis similar al anterior obteniendo la siguiente gráfica:

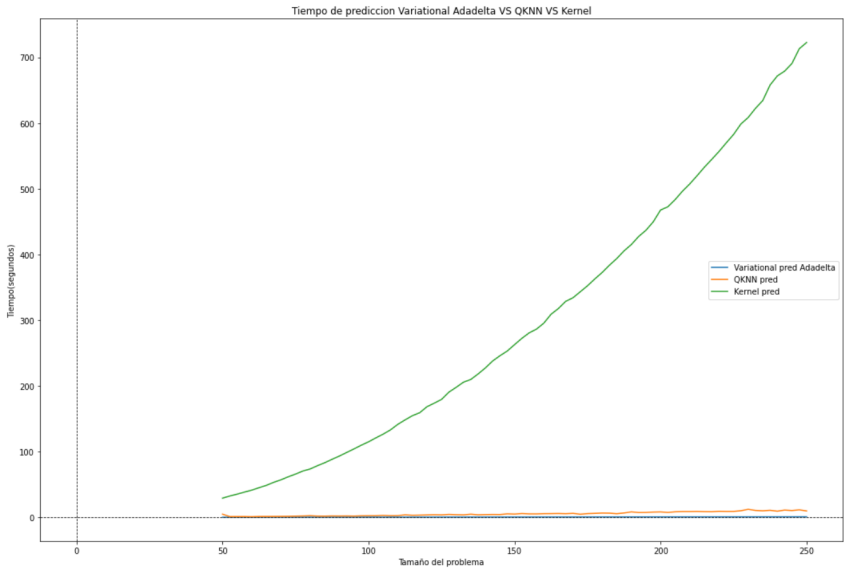


Figura 32. Gráfica comparativa del tiempo de predicción escalado de los modelos Kernel, QKNN y QVC.

Aquí de nuevo ocurre que el modelo basado en Kernel escala sustancialmente peor que los otros dos modelos. Sin embargo, en esta ocasión sí que se aprecian diferencias entre el modelo cuántico de K-vecinos y el modelo variacional cuántico, donde el primero escala ligeramente peor. Si eliminamos el modelo Kernel de la gráfica anterior, se obtiene la siguiente gráfica:

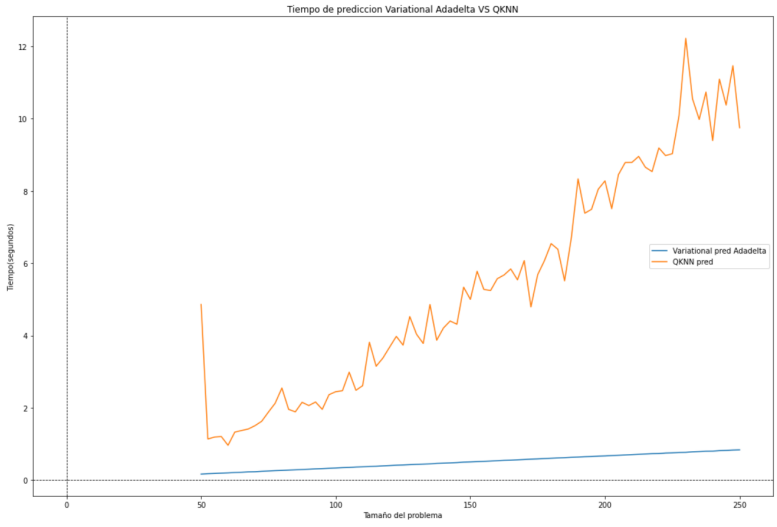


Figura 33. Gráfica comparativa del tiempo de predicción escalado de los modelos QKNN y QVC.

En esta se ve con mayor claridad las diferencias entre ambos donde, como se observó anteriormente, el modelo cuántico de K-Vecinos escala ligeramente peor. En este caso, además, se aprecia que es el modelo QKNN el que posee picos o variaciones en su gráfica. Esto es debido a lo mencionado en la sección "Preparación de los modelos", donde el problema no es divisible entre el número de cores utilizado, obteniendo mejores y peores rendimientos con diferentes tamaños del problema. Además, aquí la gráfica del modelo cuántico variacional es completamente lineal ya que el componente aleatorio del modelo existe únicamente durante el entrenamiento mientras que, a la hora de predecir, este debe aplicar los parámetros entrenados en la fase anterior.

Es importante recalcar que la versión cuántica de K-Vecinos está paralelizada mientras que el modelo variacional cuántico se ejecuta en serie haciendo que requiera menos recursos para su ejecución.

2. RESULTADOS CONSEGUIDOS

Justificación 2021

A lo largo de la anualidad actual, se han logrado alcanzar los resultados esperados según la memoria del proyecto. Estos objetivos están asociados a los cuatro primeros objetivos específicos definidos en dicha memoria, donde se ha alcanzado en su totalidad la relativa al primero, y de forma parcial los tres siguientes.

OE1. Identificación y planteamiento matemático de los problemas susceptibles de ser abordados mediante técnicas de computación cuántica:
Tal y como se ha indicado previamente, el OE1 se ha satisfecho completamente donde, tras analizar los diferentes retos que se relacionan con la industria para tres campos de estudio; definir tres subproblemas abordables tanto a través de técnicas de computación clásica como cuántica e identificar técnicas tanto clásicas como cuánticas para abordar cada uno de los tres subproblemas.
Con respecto a los objetivos específicos en proceso de consecución (OE2, OE3 y OE4), se han logrado avances parciales, si bien no la consecución completa de estos, dado que su desarrollo está ligado a paquetes de trabajo que no está planificado cerrar en la anualidad 2021, sino que en las siguientes.
Por un lado, el objetivo específico "**OE2: Diseño de los algoritmos basados en computación clásica que puedan solucionar los problemas identificados**" está estrechamente ligado con el desarrollo del paquete de trabajo "PT2: Análisis, diseño y desarrollo de algoritmos basados en computación clásica".

En lo relativo al "**OE3: Diseño de los algoritmos cuánticos necesarios para abordar los problemas identificados**", ligado al paquete de trabajo "PT3: Análisis, diseño y desarrollo de algoritmos basados en computación cuántica".

Finalmente, el último objetivo específico abordado a lo largo de esta anualidad es el "**OE4: Creación del *framework* de simulación cuántica**", que en este caso, se encuentra asociado al paquete de trabajo "PT4: *Framework* de simulación cuántica".

La completitud de estos tres objetivos específicos se llevará a cabo a lo largo de las siguientes anualidades, partiendo como base de lo indicado previamente.

Los resultados generados hasta el momento y que se pretende obtener durante el curso del proyecto se espera contribuyan a un mayor entendimiento de las posibilidades de la computación cuántica en ejemplos reales, concretamente ejemplos de la industria regional, complementando proyectos de investigación recientes, en los que CTIC ha tomado parte activa transfiriendo tecnología a clientes del ámbito industrial y confirmando así el compromiso de CTIC con la resolución de problemas de sistemas productivos y de digitalización de la industria.
A nivel interno, ALCATRAZ está totalmente alineado con la Estrategia del Centro, tanto a nivel tecnológico (promoviendo avances en la línea de Computación Cuántica) como respecto a su contribución a la Misión/Visión del Centro (generación de valor en empresas, apoyo a la industria regional, alineación con políticas nacionales y europeas). En base a ello, ALCATRAZ es uno de los proyectos estratégicos que CTIC-Centro Tecnológico ha lanzado en el año 2021 para vertebrar el I+D asociado al despliegue de su *roadmap* tecnológico.

Justificación 2022

A lo largo de la anualidad actual, se ha trabajado para alcanzar los resultados esperados según lo descrito en la memoria del proyecto:

- Una mejor **base de conocimiento** en la materia, que aborde los principales retos a los que se enfrenta el sector industrial desde un punto de vista matemático. Para ello, se focalizarán los retos en los contextos de la clasificación, optimización y simulación, por ser tres grandes tipos de problemas para los que se ha demostrado que un enfoque cuántico es adecuado para su resolución.

- Un **conjunto de algoritmos cuánticos** diseñados para solventar los problemas planteados.

- Un **conjunto de algoritmos de IA** que servirán para validar los resultados obtenidos con los algoritmos cuánticos.

- Un avanzado ***framework* tecnológico** capaz de ejecutar los algoritmos cuánticos diseñados rápida y eficientemente. Será una combinación de elementos de hardware y software que permitirá realizar experimentos cuánticos desde arquitecturas de computación tradicionales.

Se han realizado grandes avances respecto al primer punto sobre la generación de una mejor base de conocimiento matemático sobre los principales retos que afronta el sector industrial. Ya en la anualidad 2021, se hizo un extenso estudio sobre los subproblemas matemáticos de interés: clasificación, optimización y simulación para en la anualidad 2022 ahondar en los casos de uso específicos que se podían asociar a estos subproblemas: anticipación a ciberataques, optimización de rutas y diseño de nuevos productos.

El segundo resultado, la obtención de un conjunto de algoritmos cuánticos para solventar los problemas planteados, se ha cumplido con éxito ya que se han obtenido tres algoritmos cuánticos para el subproblema de clasificación, un algoritmo cuántico para el subproblema de optimización y un algoritmo cuántico para el subproblema de simulación.

El tercer resultado, la obtención de un conjunto de algoritmos de IA para validar los resultados de los algoritmos cuánticos está en curso, puesto que se ha obtenido el conjunto de algoritmos de IA, pero todas las actividades de validación de resultados caen en la anualidad 2023.

El cuarto resultado, el framework tecnológico capaz de ejecutar los algoritmos cuánticos diseñados rápida y eficientemente, se ha cumplido. Queda pendiente una continua monitorización del sistema durante la anualidad 2023 para garantizar su éxito.

Estos resultados están asociados a los siguientes objetivos específicos definidos en la memoria:

- **OE2. Diseño de los algoritmos basados en computación clásica que puedan solucionar los problemas identificados:**

El OE2, parcialmente cubierto en la anualidad 2021, se ha satisfecho completamente en esta anualidad 2022 donde los algoritmos de computación clásica previamente identificados han sido validados. Para cada uno de los tres problemas, se han identificado un número mayor de posibles modelos a aplicar con respecto a la anualidad previa, habiéndolos comparado experimentalmente para definir cuáles eran los idóneos en los casos de aplicación particulares.

- **OE3. Diseño de los algoritmos cuánticos necesarios para abordar los problemas identificados:**

El OE3, también estaba parcialmente cubierto en la anualidad 2021, y también se ha satisfecho completamente en esta anualidad 2022 donde los algoritmos de computación cuántica previamente identificados han sido validados, a través de la implementación real de estos algoritmos en un ejemplo simple para verificar su rendimiento y viabilidad.

- **OE4. Creación del *framework* de simulación cuántica:**

El OE4 queda cubierto en esta anualidad 2022, puesto que la única tarea pendiente en el proyecto es la monitorización del *framework* que se realizará en la anualidad 2023.

- **OE5. Realización de pruebas de concepto en el *framework* de simulación cuántica:**

Con respecto al objetivo específico en proceso de consecución OE5, se han logrado avances parciales, si bien su consecución completa, llegará en la próxima anualidad 2023.

Los resultados generados hasta el momento y los que se pretenden obtener durante el curso del proyecto contribuyen a un mayor entendimiento de las posibilidades de la computación cuántica en ejemplos reales, concretamente ejemplos de la industria regional, complementando proyectos de investigación recientes, en los que CTIC ha tomado parte activa transfiriendo tecnología a clientes del ámbito industrial y confirmando así el compromiso de CTIC con la resolución de problemas de sistemas productivos y de digitalización de la industria.
A nivel interno, ALCATRAZ está intrínsecamente alineado con la Estrategia del Centro, tanto a nivel tecnológico (promoviendo avances en la línea de Computación Cuántica) como respecto a su contribución a la Misión y la Visión del Centro (generación de valor en empresas, apoyo a la industria regional, alineación con políticas nacionales y europeas). En base a ello, ALCATRAZ es uno de los proyectos estratégicos que CTIC Centro Tecnológico lanzó en el año 2021 para vertebrar el I+D asociado al despliegue de su *roadmap* tecnológico en torno a la cadena de valor del dato.

Justificación 2023

A lo largo de la última anualidad del proyecto, se ha trabajado para alcanzar los resultados esperados según lo descrito en la memoria del proyecto:

- Una **base de conocimiento** ampliada en la materia de simulación cuántica gracias al trabajo ejecutado en la anualidad 2023 que aplicaba esta tecnología emergente en ejemplos reales de la industria asturiana.
- Un ***framework* tecnológico** optimizado y ajustado para ser capaz de implementar algoritmos cuánticos de manera rápida y eficiente.
- Un **conjunto de algoritmos cuánticos** probados en el *framework* para los casos de uso planteados.
- Una **comparación** entre los resultados obtenidos en un entorno de **simulación cuántica y** con **computación clásica**.

Estos resultados están asociados a los siguientes objetivos científico-técnicos específicos definidos en la memoria:

- **OE4. Creación del *framework* de simulación cuántica:**

El OE4, ya cumplido en la anualidad 2022, se ha optimizado durante la anualidad 2023 al solucionar las incidencias ocasionadas durante la tarea T4.3 Monitorización del *framework*.

- **OE5. Realización de pruebas de concepto en el *framework* de simulación cuántica:**

El OE5 se ha cubierto en su totalidad en la anualidad 2023, puesto que los algoritmos cuánticos diseñados se han probado en el *framework* de simulación cuántico para los tres casos de uso identificados.

- **OE6. Validación de la viabilidad de las soluciones cuánticas encontradas:**

El OE6 se ha cumplido en la anualidad 2023 con la comparativa realizada entre los resultados con los algoritmos cuánticos y clásicos.

Los resultados generados en el proyecto ALCATRAZ contribuyen a una mayor comprensión de las posibilidades de la computación cuántica aplicada a ejemplos reales, concretamente ejemplos de la industria regional, aportando valor añadido a otros proyectos de investigación en los que CTIC ha transferido tecnología a clientes del sector industrial, confirmando así su compromiso con la resolución de problemas enmarcados en el concepto de Industria 5.0, uno de los ámbitos de actuación clave para el Centro.

A nivel interno, ALCATRAZ, intrínsecamente alineado con la estrategia del Centro, ha servido para afianzar la línea de investigación de Computación Cuántica, lo que ha servido a CTIC para posicionarse a nivel nacional en esta tecnología y, en colaboración con otros Centros Tecnológicos nacionales, liderar una propuesta continuista en esta línea de investigación.