Desarrollo de Automático mediante Python para Optimización de Redes de Telecomunicaciones

[1]Luis Gabriel Mateo Mejía, [2]Karla Georgina Olivo Bernal.
[1]Doctorante de Sistemas Computacionales de la Universidad DaVinci
Orcid: 0000-0001-8289-1377
gabriel.mm@purhepecha.tecnm.mx
Cherán, Michoacán, México
[2]Docente de ISIC del Instituto Tecnológico Superior Purépecha
Orcid: 0000-0003-4673-300X
karla.ob@purhepecha.tecnm.mx
Cherán, Michoacán, México

Recibido*:22 de octubre de 2025*
Aceptado*: 8 de diciembre de 2025*

RESUMEN.

El presente artículo se realiza un análisis bajo el método cuasi experimental, una generalización del código de aprendizaje automático para grafos, que optimiza y simula casos concretos de redes neuronales artificiales, como es una red de telecomunicaciones de tipo dinámica, dando como resultado una carga de red estable y continua para un servicio de ancho de banda pre configurado. Conclusivamente el ML es factible completamente en la minería de grafos y se integra a la IA bajo el esquema de una computación binaria que optimiza su trabajo.

PALABRAS CLAVE: Aprendizaje automático, teoría de grafos, optimización y simulación de redes.

ABSTRACT.

This article aims to analyze, using a quasi-experimental method, a generalization of machine learning code for graphs that optimizes and simulates specific cases of artificial neural networks, such as a dynamic telecommunications network. This results in a stable and continuous network load for a pre-configured bandwidth service. In conclusion, machine learning is entirely feasible for graph mining and integrates with AI under a binary computation scheme that optimizes its operation at millisecond speeds.

KEY WORDS: Machine learning, graph theory, telecommunications network, network optimization and simulation.

1. INTRODUCCIÓN

A menudo nos preguntamos cómo es posible que la IA pueda optimizar una red de telecomunicaciones bajo ciertos requisitos de demanda. Más aún, nos enfrentamos a una pregunta más profunda a nivel de programación en sistemas, ¿de qué manera se logra el aprendizaje automático bajo parámetros de grafos? Esto nos lleva directamente a la predicción, optimización o detección de anomalías en una red tanto de telecomunicaciones como de los sistemas web. En algún momento del desarrollo de la IA se ha recurrido al aprendizaje automático como medio o estructura que permite lograr y generar nuevos conocimientos. De igual manera, la minería de

grafos es una aplicación directa del desarrollo de software que vincula sustancialmente la aplicación de las ciencias básicas de la ingeniería y sus problemas de desarrollo tecnológico. (Ponce. 2010). El desarrollo de la minería de grafos se ha constituido como una herramienta muy concreta que viene a genera y desarrolla software para la web y las aplicaciones del internet a nivel científico. Interdisciplinario y social. (Caicedo. 2010). Por tal motivo, se pretende demostrar la factibilidad hipotética de entrenar modelos o algoritmos de aprendizaje automático bajo ciertos parámetros concretos como es el hecho de contar con una red de telecomunicaciones específica. Así logramos demostrar su factibilidad que va de lo particular a lo general para este tipo de bienes no tangibles que manejan y controlan el sector industrial y social.

1.1 MARCO TEÓRICO

A manera de teoría de base, podemos señalar que GM, (la minería de grafos), surge en la intersección de la teoría de grafos, la estadística y la vinculación impulsada por la IA para analizar datos complejos en la década de 1990. Hasta la fecha combina GNNs, escalabilidad y aplicaciones multimodales.

Dentro del esquema de conexión para este pipeline, (diagrama de flujo metodológico), podemos considerar los siguientes programas de codificación como una secuencia para encontrar la solución al objetivo de optimización y simulación en IA mediante la teoría de grafos. a) Las GNNs, permiten construir el aprendizaje profundo en la representación de grafos, no operan en datos estructurados o secuencias lineales, sino que manejan la complejidad no euclidiana de los grafos. b) Seguido del modelo ARIMA, que sigue un ciclo iterativo de identificación, estimación y diagnostico que comprende un proceso pasado, esto permite proponer un patrón de comportamiento y conducta de los datos respecto a la predicción de lo que puede suceden en el transcurso del tiempo futuro. c) Seguimos con el modelo PROPHET que analiza y considera los datos de las series temporales con el objetivo de modelar la tendencia, la estacionalidad, (anual, semanal, diaria, etc), y los efectos de días festivos. Incluye los años de historia e incorpora factores externos como promociones y/o ventas. (Ma.,y Tang. 2021). Los tres incisos anteriores describen los criterios de validación para que concretizan la capacidad de la teoría de grafos para predecir, optimizar y simular procesos de ingeniería. (Hamilton. 2020).

Los antecedentes de GM se dan en las bases matemáticas para modelar redes como lo explican en sus artículos sobre desarrollo de la ciencia de la teoría de grafos Erdős y Rényi, en 1959. En los años 90, los grafos representan de forma natural el hipertextos y las redes sociales, permitiendo a los algoritmos tempranos comenzar su proceso de ML con la clasificación, clustering y detección de comunidades. Finalmente para 2010 se da una explosión de aplicaciones tanto en bioinformática como en ciberseguridad, dando pauta a la extracción de patrones significativos de estructuras grafales masivas, evolucionando desde algoritmos exactos a métodos escalables y probabilísticos

El actual estado del arte de la minería de grafos se sitúa en los modelos dominantes como GNNs y GraphSAGE, GAT, la explicabilidad y la relación heterogénea: R-GNC, así como su escalabilidad para o Graph coarsening y sistemas distribuidos (PyG, DGL). Es importante señalar que se han desarrollado tareas avanzadas como es el caso de Link Predictión, Anormal y detección y Generación de grafos, bajo modelos de lenguaje y privacidad, así como la inserción en el nuevo paradigma exploratorio de Quantum graph algoritms. Frameworks como PyTorch Geometric y Deep Graph Library lideran implementación. (Barabási. 2012).

Entre la IA clásica y el aprendizaje automático existen cambios fundamentales en el paradigma que permite procesar la información, pasando del simple uso causal de bits al uso controlado de los mismos. De igual manera, el paradigma al cambio de la computación cuántica contiene cambios estructurales para procesar la información

y resolver problemas, pasando de la computación binaria a la representación simultanea de 0 y 1, lo que supone la superposición y el entrelazamiento de partículas electrónicas y magnéticas a distancia. De manera particular, la IA clásica automatiza bajo operaciones de secuencia lineal, utilizando estructuras repetitivas y limitación en volumen de datos, la IA clásica no aprende ni modifica su comportamiento por experiencia debido a que sus datos son de tipo no estructurados. (IBM. 2025). De aquí que el aprendizaje automático, ML, construye modelos a partir de ejemplos que mejoran el desempeño y la experiencia, aun fuera de la programación explicita. Esto permite interpretar patrones y tomar decisiones. Finalmente, llegamos al aprendizaje automático cuántico, el cual usa qubits, lo que permite procesar millones de operaciones en paralelo, ofreciendo una optimización y aceleración de mejora para los algoritmos de IA y de ML. (Benítez. 2014).

El aprendizaje automático cuántico combina la mecánica cuántica con técnicas de machine learning para optimizar algoritmos, procesar grandes volúmenes de datos a mayor velocidad y resolver problemas complejos fuera del alcance de la IA clásica. Puede hacer que la IA sea más precisa y confiable, por ejemplo, filtrando mejor la información y mejorando la interpretación de datos ambivalentes. El salto ocurre al pasar de sistemas basados en reglas fijas (IA clásica) a máquinas que aprenden de datos (machine learning) y luego a la integración con la computación cuántica, que amplifica exponencialmente la capacidad de cálculo y mejora la eficiencia y alcance de las técnicas de IA. (Chollet. 2018).

Al hablar de GML nos encontramos en el área de la minería de datos y la minería aplicada a la integración del aprendizaje automático. En este sentido, las librerías de Python que cada vez se enriquecen y se agregan en gran medida son: pandas, elementree, matplotlib, torch-tensor, sklearn, networkx, graph, openpyxl, prophet, pulp, entre otras de usual acoplamiento y visualización. (Chollet. 2018).

Ahora bien, el aprendizaje profundo en GML es el desarrollo del aprendizaje para extraer características de los grafos, predecir etiquetas de los nodos y nuevas conexiones. (Bratanic. 2024). Ello implica un trabajo previo de algoritmos en redes inferidas, que busca los nodos críticos, así como la identificación de sus similares, no sin antes considerar las comunidades de los mismos y sus relaciones indirectas. Ahora bien, el lenguaje de consulta de grafos es por excelencia otro tipo de trabajo que permite generar y desarrollar la estructura del grafo, aquí es donde las librerías como Neo4j, Cytoscape y otras, son de gran utilidad. En el caso del lenguaje de consulta es necesario generar los algoritmos que permitan aprender a identificar patrones en los grafos, revisar las conexiones que son de tipo transversales, agregar y quitar datos, y explorar su conducta mediante análisis estadísticos. (Bratanic. 2024). Así como también terminamos con un desarrollo de la IA generativa que permite en todo momento construir y modelar grafos. Esto permite aprender a identificar los patrones y relaciones entre sus puntos, la consecución del lenguaje DOT y la generación de su matriz de adyacencia, genera la descripción de la estructura del grafo y poder importar los datos de una base de datos especializada para grafos. (Russell. 2004).

La inteligencia artificial se ha destacado por desarrollar modelos en código y algoritmo que pueden ser entrenados en línea bajo los parámetros de un caso particular y así generar pronostico, simulación y predicción. (Soria. 2022). Se visualiza a futuro que la IA contara con estos mismos hallazgos de IA en entrenamiento de modelos pero en la computación cuántica, como lo muestra (IBM-QUANTUM. 2025), en la imagen a continuación:

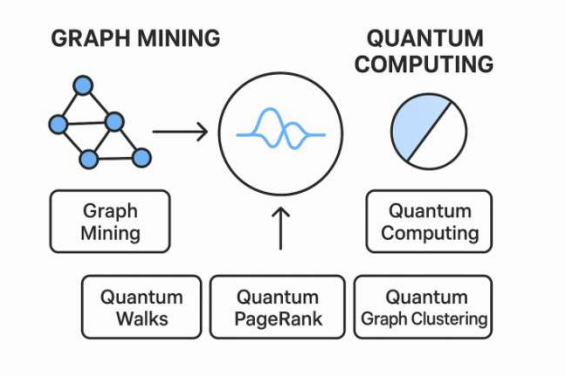


Imagen 7. Relación de minería de grafos y cómputo cuántico.

La computación cuántica permite resolver problemas complejos de grafos, pues ya de por si la teoría de grafos es una teoría matemática compleja. Son conocidos los problemas de optimización, como la ruta más cota, el conjunto independiente máximo, la coloración del grafos, el problema de corte para máximos conjuntos y los algoritmos de grafos, como el del Grover, para buscar datos; el de optimización, (Quantum Approximate Optimization Algorithm) y el algoritmo de recocido cuántico (Quantum Annealing, usado en máquinas D-Wave. (Yanofsky. 2008). El análisis de redes permite por otra parte modelar circuitos cuánticos, (computación hibrida y cuántica para manejo de arquitectura con Qubits), y revisar su conectividad, entrelazamiento, su estado cuántico, su corrección de errores, para simular fenómenos cuánticos. (Yanofsky. 2008).

1.3 PLANTEAMIENTO DEL PROBLEMA

En esta investigación se propone como objetivo general demostrar que la programación binaria llevada a la inteligencia artificial clásica permite integrar la teoría de grafos en la minería de grafos con las funcionalidades de optimización, simulación y predicción de errores en sus distintas aplicaciones.

Nuestra hipótesis es básica, dado un grafo dinámico de una red de telecomunicaciones concreta, generalizar su aplicabilidad dentro de los parámetros de optimización, prevención de anomalías y generación de una red neuronal de aprendizaje profundo.

Como objetivos específicos se analizará la capacidad de generar una red de telecomunicaciones en su representación de un grafo dinámico, siguiendo la ciencia de redes y llevando este programa a la codificación web para integrarlo a IA en el aprendizaje automático. Con este proceso se pretende mostrar y demostrar que es posible y factible evaluar la red, optimizarla y cuidarla ante fallos o anomalías de conexión. (Ponce. 2010).

De esta forma, las distintas IA como Gemini, Grok o ChatGpt, nos pueden generar una gráfica en donde visualicemos un problema de conectividad de una red de telecomunicaciones de 150 terminales fijas y 150 intermitentes con horas pico de 11 am a 15 horas, como se muestra a continuación: En la siguiente imagen 1 se muestra el grafo de una red bajo las características del modelo-problema para posteriormente trabajar su estructura de datos en el software Gephi.

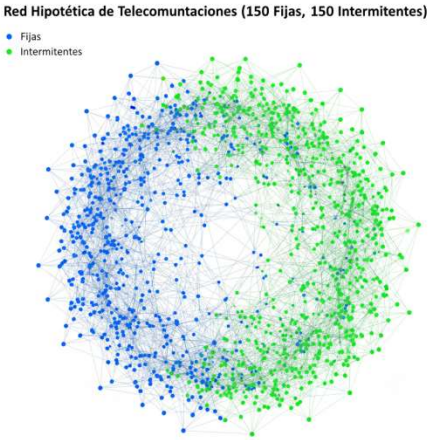


Imagen 1. Red hipotética de telecomunicaciones con horas pico.

Cabe señalar que IA como Perplexity no generan la visualización de grafos. Sin embargo, una red no hipotética queda construida en Gephi, como se muestra en la siguiente imagen 2. A su vez, al ser un grafo dinámico, muestra una de las horas pico, las 13 hrs., con sus estadísticas y métricas para hora pico de conectividad. (Gephi. 2025). En la imagen 2 observamos el grafo dinámico de la red de telecomunicaciones bajo las características prefiguradas, haciendo hincapié en la línea de tiempo, especialmente de las 13 horas, donde la intensidad de carga es mayor.

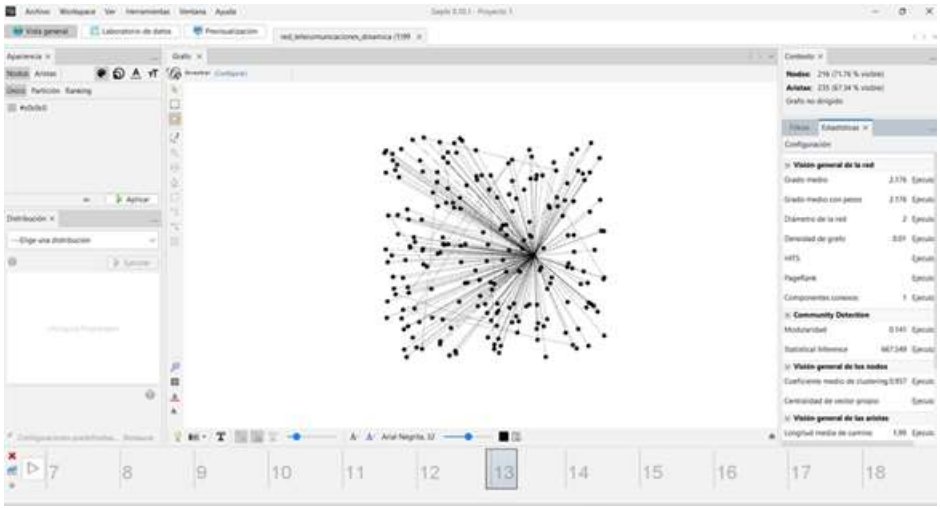


Imagen 2. Red no hipotética de telecomunicaciones.

Además, el software Gephi puede importar archivos de base de datos csv o xls, pero su producto no lo puede visualizar Gemini u otras IA como Grok. Por lo que se pretende concretizar la conectividad de la red en Intermitente, Fija y Hr-Pico bajo la vinculación de celdas y columnas, como se muestra en la imagen 3-4. De forma particular, la imagen 3, nos permite administrar la base de datos del grafo de la red de telecomunicaciones bajo la estructura de filas y columnas, lo que facilita la extracción y manejo de la información.

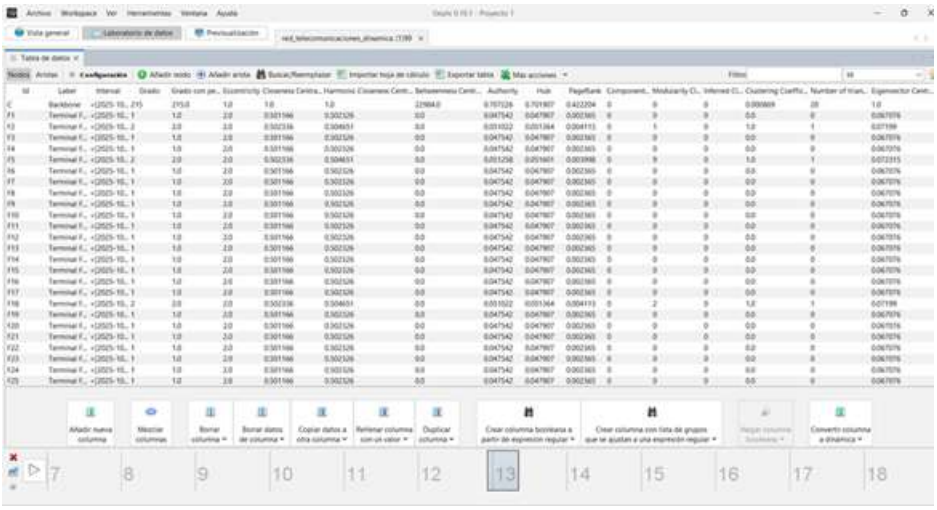


Imagen 3. Formulación de la conectividad Intermitente y Fija.

Finalmente, el manejo de Gephi permite visualizar el grafo desde la conectividad de todas sus horas de uso, de forma específica las 13 horas, que es una hora pico, como se observa en la imagen 4 a continuación. Este esquema nos regala la configuración de las particularidades del grafo bajo los elementos de nodos, aristas y conectividad.

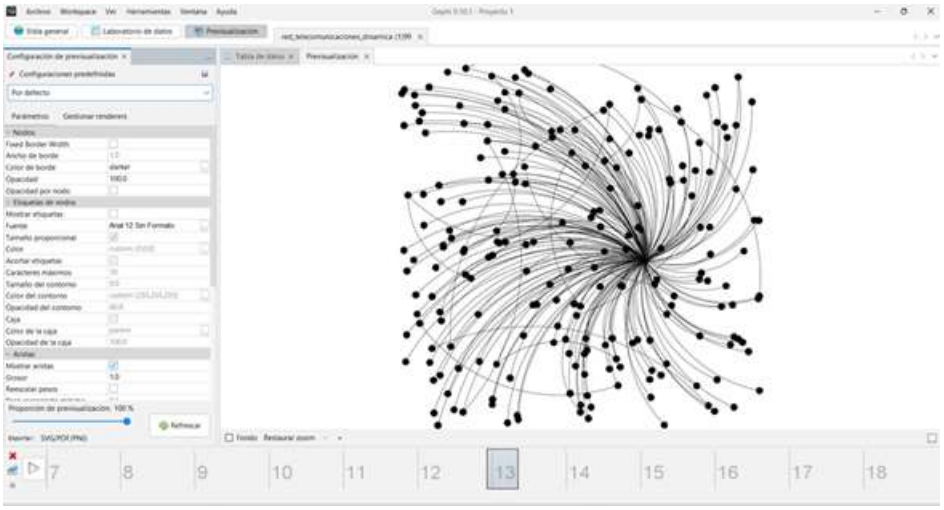


Imagen 4. Visualización de la conectividad y hora pico.

De igual manera, se muestran las estadísticas de Gephi, coincidentes con la programación de Python en Colab, y que se muestran en la siguiente imagen 5 resumida:

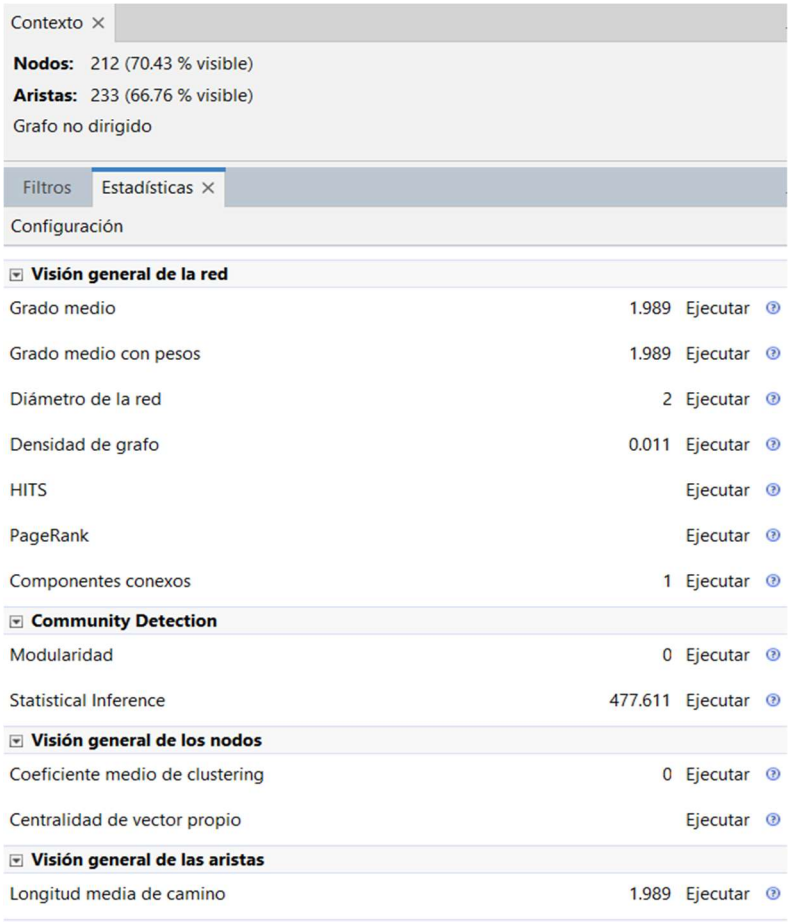


Imagen 5. Métricas y estadísticas de Gephi, coincidentes con Colab.

Por su parte en la versión de Colab, las métricas coinciden con las de Gephi, destacando su semejanza en aristas, nodos, diámetro de red, centralidad y densidad del grafo. Es importante resaltar que la planeación del ML requiere el encuadre del problema que será guía y modelo para aplicar los algoritmos de IA, por tal motivo, se considera un modelo de prototipo tanto de código como de software para estructurar los conocimientos y temas de base en la minería de grafos. (Soria. 2022).

De forma inductiva, a través de un caso práctico, podemos generalizar los aspectos esenciales de los logaritmos para aprendizaje automático de grafos. En este caso, buscamos optimizar el ancho de banda de una red de telecomunicaciones, en donde el grafo contiene los nodos que representan los routers y las aristas, las conexiones físicas, con las características de demanda de tráfico en horas pico y capacidad. Como objetivo tenemos asignar un ancho de banda r_v a cada router para maximizar el throughput, sujeto a un límite total de ancho de banda. En este caso el algoritmo usa GNN´s para generar embeddings $h_v^{(k)}$ que capturen la demanda y las conexiones. Para grafos dinámicos, con demanda cambiante, usamos LSTM[1] para modelar $h_v^{(t)}$. Predecimos así r_v con una red

[1] LSTM (Long Short-Term Memory) es un tipo de arquitectura de red neuronal recurrente (RNN) diseñada para manejar secuencias de datos y resolver problemas como el desvanecimiento del gradiente en redes tradicionales.

neuronal o política de RL, optimizamos así la función objetivo: max$\sum throughput(r_v)$, sujeto a $\sum(rv) \leq R$, lo que resulta en una asignación optima de ancho de banda que minimiza la congestión.

2. METODOLOGÍA.

Esta investigación se ubica dentro de su propósito como investigación aplicada, ya que tiene propósitos prácticos inmediatos que dependen de las ciencias básicas y sirven para transformar. En cuanto a su medio para obtener los datos es el campo de la web semántica, mismo espacio que sirve de laboratorio de datos y documenta la información. (Hernández. 2014). Por la naturaleza de los datos, su enfoque es mixto, tanto cualitativo como cuantitativo.

Por su profundidad, según el alcance de la investigación, la podemos situar en su tipo exploratoria, descriptiva, correlacional, explicativa, predictiva y aplicativa, enfocándonos en este último punto, puesto que los estudios de IA son de innovación, permiten solucionar problemas, controlar situaciones y aplicar conocimientos adquiridos. (Hernández. 2014).

En cuanto a la mayor o menor manipulación de las variables tenemos un enfoque tanto experimental como cuasi-experimental, debido a que las operaciones de GML tienen como base las operaciones de las matemáticas aplicadas. En cuanto al periodo temporal en que se realiza la investigación es de tipo longitudinal. En cuanto a su tipo de inferencia se utilizan los métodos estadísticos, deductivos, hipotético-deductivos y analíticos. En cuanto al tiempo en que se efectúa la investigación es de tipo asincrónico. (Hernández. 2014). En la siguiente imagen 6 se muestra el diagrama de flujo metodológico que se realiza en este estudio, resaltando las partes del proceso para la generalización del código de ML:

Imagen 6. Diagrama de flujo metodológico.

Los datos utilizados son 150 conexiones fijas y 150 conexiones móviles, con IP dinámica, con las siguientes características: los nodos con Id, etiqueta e intervalo. Las aristas con origen, destino, tipo: dirigida o no dirigida, Id al nodo, a la etiqueta, señalización de intervalo y peso, por el tiempo de uso y hora pico de anclaje. En cuanto a las variables medidas tenemos: a) las métricas propias de los grafos dinámicos, como es centralidad, nodos, aristas, diámetro y densidad, como se muestran en la tabla 1 arriba; b) las variables propias de las redes neuronales gráficas, como son: patrones y tendencias regulares en el tiempo, de tipo estadística predictiva, en

este caso contamos con actividad en los nodos vs el trayecto del tiempo, serie ARIMA; de igual manera contamos con la variable t-SNE, que permite visualizar la estructura local subyacente de datos de alta dimensión en dos o tres dimensiones, lo que permite representar los resultados en un diagrama de dispersión simple, para redes artificiales de grafos, GNNs; así mismo se plantea la variable anomalía, en la relación actividad de nodos vs tiempo, con detección de carga en la red anómala; por último pretendemos optimizar el recurso de la red, en su funcionalidad desgaste de energía vs tiempo.

A su vez, las herramientas de procesamiento son las siguientes: primeramente software Gephy, después pasamos a las librerías de Python para ML y para IA. Uso de Notebook de la suite de Júpiter en Google Colab para IDE y desarrollo de código Python. Terminamos con visualización de IA, en este caso Gemini. Una vez que contamos con la red y su grafo respectivo, procedemos en la metodología a procesarla en algoritmos para IA y especialmente para aprendizaje automático. En el siguiente cuadro comparativo (Tabla 1), podemos observar de una forma breve y muy resumida una comparación teórica entre los algoritmos a usar en este proyecto de evaluación de optimización y simulación, así como las aplicaciones directas en un proceso de evolución de IA clásica a ML para grafos:

Algoritmos: (Wu. 2022).	**Referencia teórica: (Han. 2012).**	**Aplicación: (Witten. 2011).**
Redes Neuronales Gráficas, (GNNs).	Graph Neural Networks: Transformación de las matrices de adyacencia y manejo de base de datos.	Clasifica nodos, grafos y predice enlaces, anomalías y genera modelos.
El algoritmo ARIMA: sigue un ciclo iterativo de Identificación, Estimación y Diagnóstico (a veces llamado Box-Jenkins Methodology, aunque ese es un proceso más amplio).	Algoritmo ARIMA (p, d, q) para Predicción de Series de Tiempo. Objetivo: Modelar una serie de tiempo univariada Yt y realizar predicciones futuras. Entrada: Una serie de tiempo Yt con T observaciones. Salida: Un modelo ARIMA (p, d, q) y predicciones futuras de Yt. Aplicar pruebas de raíz unitaria como la Prueba de Dickey-Fuller Aumentada (ADF) o la Prueba de Kwiatkowski-Phillips-Schmidt-Shin (KPSS).	Una serie estacionaria tiene media, varianza y estructura de autocorrelación constantes a lo largo del tiempo. Los modelos ARIMA requieren estacionariedad. Una vez que la serie es estacionaria (después de d diferenciaciones), analizamos las funciones de autocorrelación. Una vez que se tiene un modelo ARIMA (p, d, q) validado, se utiliza para predecir valores futuros de la serie de tiempo. El modelo utiliza los valores pasados observados y los errores pasados para generar los

		pronósticos y sus intervalos de confianza.
Algoritmo para optimización de recursos. Ford-Fulkerson (incluyendo métodos como Edmonds-Karp). Algoritmos que combinan flujo máximo con camino más corto, como la Programación Lineal o variaciones de Ford-Fulkerson. Dijkstra (para aristas con pesos no negativos), Bellman-Ford (maneja pesos negativos), Floyd-Warshall (todos los pares de caminos más cortos). Algoritmo Húngaro, o algoritmos basados en emparejamiento de peso óptimo en grafos bipartitos.	Tipo de problema que enfrenta: Flujo Máximo (Máximo transporte o distribución a través de una red con capacidades). Flujo de Costo Mínimo (Encontrar el flujo máximo con el menor costo posible). Camino Más Corto (Optimización de rutas). Asignación (Matching) (Asignar tareas a recursos uno a uno).	Maximizar la cantidad de flujo (recursos) que pasa de un origen a un destino, respetando las capacidades de los "caminos" (aristas). Minimizar el **costo** asociado al transporte/asignación de recursos. Minimizar el costo, tiempo o distancia (recurso) entre dos o más nodos. Encontrar un emparejamiento (asignación) que minimice el costo total o maximice la ganancia.

Tabla 1. Cuadro comparativo teórico con base en algoritmos de grafos y desarrollo de software.

2.1 DESARROLLO METODOLOGICO.

El planteamiento de este problema permite formular la situación hipotética en donde se pueden aplicar distintas metodologías de ML para genera IA con dicho modelo, esto requiere de la preparación de este problema en el lenguaje de la Appi, que en este caso en Python, para posteriormente visualizarlo tanto en Google Colab como en su grafo dinámico en Gephi. Posteriormente se plantea desarrollar las siguientes técnicas de ML, aplicadas a grafos: a) Optimización de recursos en la red; b) Aplicación de series de tiempo para predicción de carga en la red; c) Detección de anomalías para detectar fallos en esta red de telecomunicaciones, y por último d) Análisis del grafo, GNNs y elaboración de sus métricas, esto permite visualizar su estructura y propiedades. (Google-Colab. 2025).

Es esencial pasar el grafo y su matriz de adyacencia a código Python, para lo cual, usaremos Google-Colab, que será la notebook principal para programar los algoritmos de aprendizaje no reforzado de IA y obtener las técnicas de GML y sus resultados. Además, como se muestra en las imágenes 8-9, la generación del grafo dinámico y su generación de modelos de aprendizaje para simulación y predicción, requiere de su propia base de datos estructurados, que en nuestro caso, tomamos como ejemplo los archivos xls y csv. Además de complementar los aspectos dinámicos del grafo no dirigido, cuya estructura matemática puede representar relaciones simétricas a través de Gephi en su ventana de laboratorio de datos. Ver imagen a continuación. (Iqbal. 2010). Primeramente

procedemos a transformar el grafo de Gephi a código Python como se observa en la imagen 8 a continuación partiendo de nuestros archivos de base de datos de Gephi.

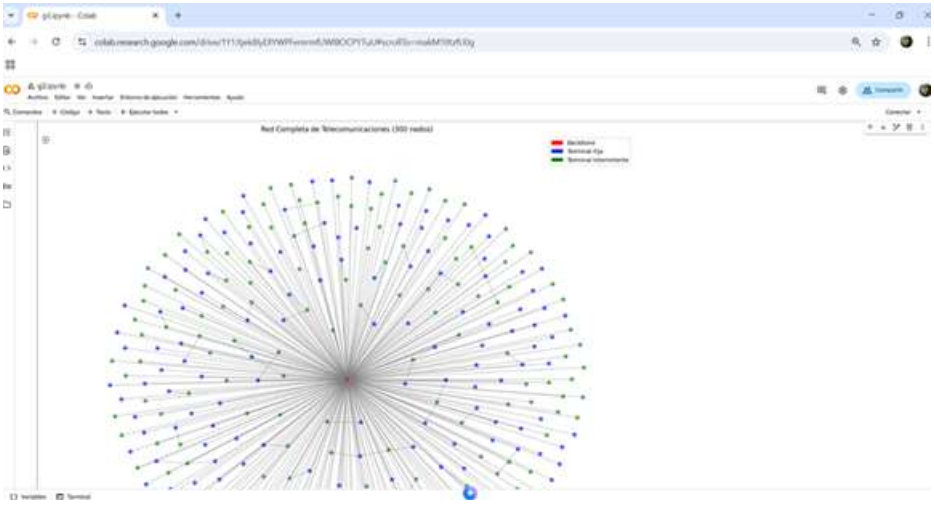


Imagen 8. Distribución de la red de telecomunicaciones.

Por otra parte, el desarrollo de los algoritmos para grafos y para desarrollo de ML, aprendizaje automático, se ha desarrollado en Python a través de diversas aplicaciones, APIs, (Application Programming Interface). Como es el caso de la interfaz Jupyter Notebook, que en Google Colab funciona de forma gratuita y no requiere instalar la suite de Acanocda Distribution, que es el IDE, entorno de desarrollo propio, como se ve en las imágenes 8 y 9, en adelante. (Bonaccorso. 2018).

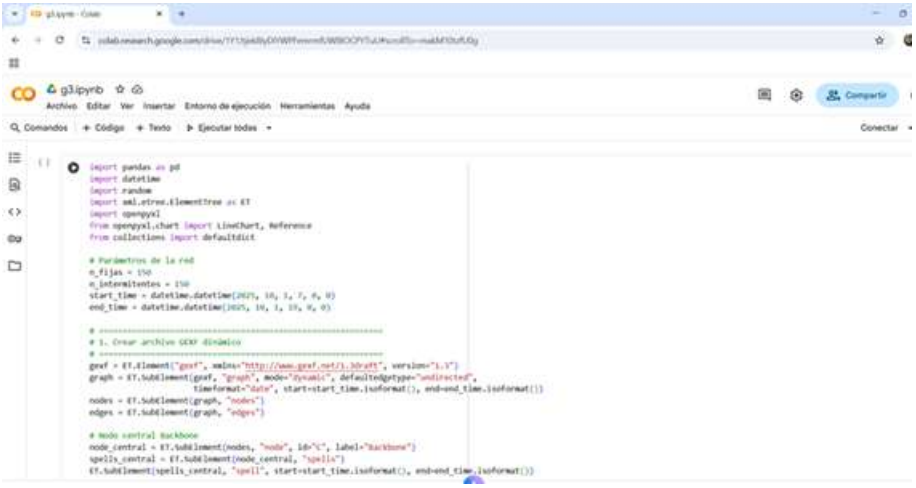


Imagen 9. Desarrollo de código para la red de telecomunicaciones.

De acuerdo a la programación y análisis del grafo dinámico, observamos que su longitud media de camino de la red es de 1.99 unidades, su diámetro es 2, y su densidad es de 0.0098 unidades, esto permite ser optimizado en un 50%. Demostrando que tiene espacios de distancia euclidiana y demostrando que entre nodo y nodo hay

la conexión a dos nodos más. Finalmente podemos observar que el coeficiente de centralidad de Clustering promedio es de 0.2835 unidades.

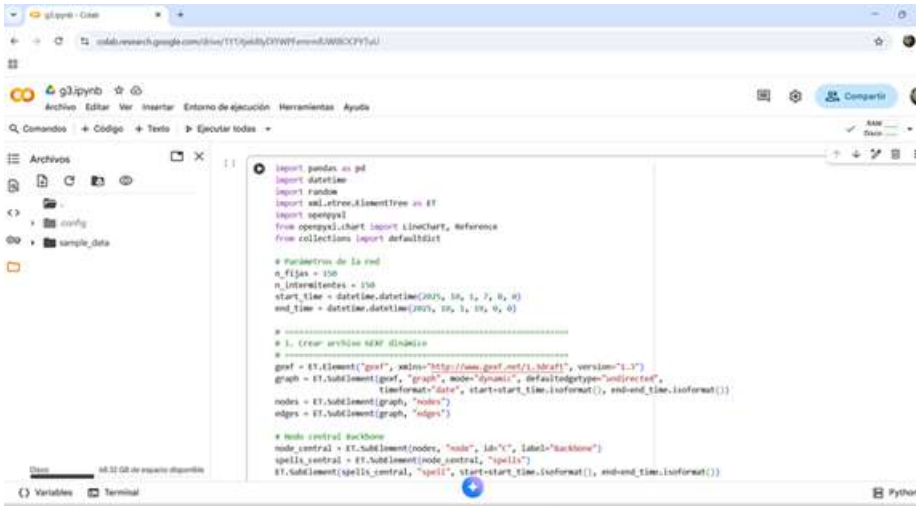


Imagen 10. Integración de pandas y librerías para ML en la codificación de la red de telecomunicaciones.

De suma importancia visualizar la red en la hora pico y su comportamiento en una gráfica de cantidad de conexiones vs horas de uso, ver la imagen 11 a continuación:

Imagen 11. Actividad de la red en hora pico.

El grafo interactivo generado por Colab con las librerías de IA y Graphics, similar al generado en Gephi, se muestra a continuación en la Imagen 12:

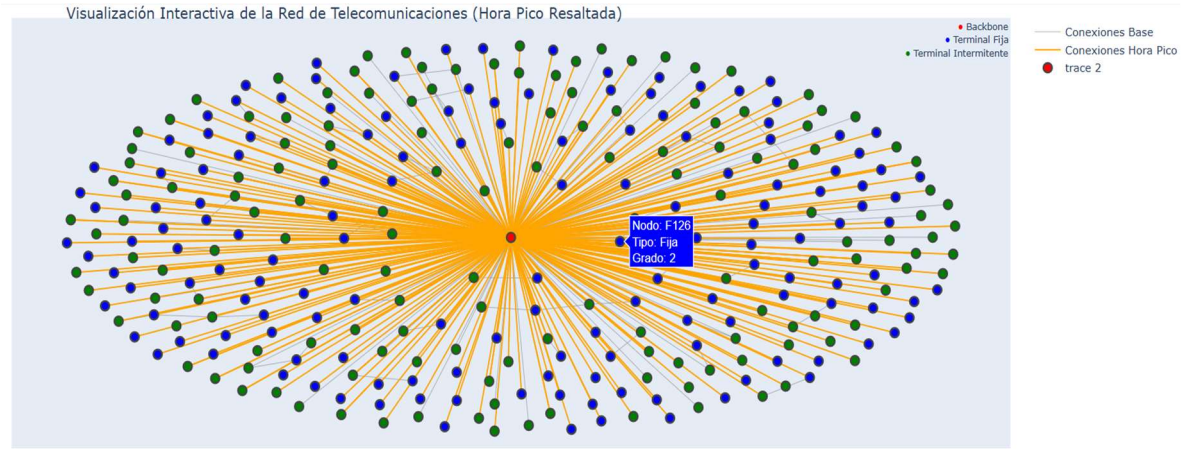


Imagen 12. Grafo dinámico de Colab.

Estadísticas y métricas de grafo: Red de telecomunicaciones para una empresa de 300 conexiones a internet por pc:

	Metric	Value
0	Number of Nodes	301
1	Number of Edges	350
2	Average Degree	2.33
3	Graph Density	0.0078
4	Number of Connected Components	1
5	Backbone Degree Centrality ('C')	None
6	Backbone Betweenness Centrality ('C')	None
7	Backbone Closeness Centrality ('C')	None
8	Average Clustering Coefficient	None
9	Graph Diameter	None
10	Average Shortest Path Length	None

Imagen 13. Métricas del grafo dinámico de la red de telecomunicaciones, (300 conexiones + hr. Pico) correspondientes al programa Gephi.

Vemos que las métricas resultantes de la tabla 3 de Gephi son congruentes con la descomposición en el código Pyhton, que resulta validado por Google Colab mediante la siguiente imagen 14.

```
Número total de nodos: 301
Número total de aristas: 350
Grado promedio de los nodos: 2.33

Distribución de grados:
  Grado 1: 211 nodos
  Grado 2: 78 nodos
  Grado 3: 11 nodos
  Grado 300: 1 nodos

Número de componentes conectados: 1
```

Imagen 14. Métricas del grafo dinámico de la red de telecomunicaciones, (300 conexiones + hr. Pico) correspondientes al programa Google Colab.

3. RESULTADOS.

A su vez, aplicando las técnicas de GML a este grafo, se observa el entrenamiento de los siguientes algoritmos y la consecución de las capacidades predictivas bajo sus procesos. (Even. 2012).

a) *Subred e Interpretación y métricas de GNNs:* son un tipo de modelo de aprendizaje profundo que trabaja con datos estructurados en forma de grafos. (Nodos y aristas que contienen entidades y sus relaciones). (Wu, L, Cui, P, Pei, J, Zhao, L. 2022). En cuanto a su algoritmo, maneja redes neuronales con datos no euclidianos, no tenemos cuadriculas o secuencias de texto. Su aprendizaje es con base en representaciones, (embeddings) de nodos, aristas, etc. Son muy versátiles en la predicción de la propagación de la información que cuya difusión se hace en la web. Recordemos que los componentes a llevar a este algoritmo son: el grafo, sus características de nodos, su vecindad, y su agregación, su actualización y sus capas. (Wu, L, Cui, P, Pei, J, Zhao, L. 2022). Este algoritmo se basa en el principio de propagación de mensajes (message passing), donde los nodos interactúan información con sus vecinos a lo largo de varias capas, (iteraciones), aprendiendo las representaciones del grafo. Su proceso es: inicialización, propagación del mensaje, (agregación y actualización), iteraciones, y lectura final, (clasifica nodos y predice enlaces). Finalmente, entrenamiento, supervisando y optimizando parámetros.

Dentro de sus variantes tenemos: GCN, Graph Convulutional Network; veamos su fórmula:

Fórmula: $h_v^{(k)} = \sigma\left(\sum_{u\in\mathcal{N}(v)\cup\{v\}} \frac{1}{\sqrt{d_v d_u}} W h_u^{(k-1)}\right)$, donde d_v es el grado del nodo v.

Imagen 15. Fórmula matemática para una red convulcional.

GraphSage, como permite generalización, puede ser una limitación al generar grafos no vistos; GAT, cuya fórmula es:

Fórmula: $h_v^{(k)} = \sigma\left(\sum_{u\in\mathcal{N}(v)} \alpha_{vu} W h_u^{(k-1)}\right)$, donde α_{vu} es un coeficiente de atención.

Imagen 16. Fórmula matemática para la red GAT total.

Graph Attention Network; y MPNN, Message Passing Neural Networks. Sus limitaciones son sobre-alisado, (oversmoothing), muchas capas indistinguibles; escalabilidad y dependencia de la calidad de las características iniciales. En la práctica se implementan con frameworks como PyTorch Geometric y DGL, un pseudocódigo simplificado sería el siguiente para generación de métricas tipo GNNs:

python Collapse Wrap Run Copy

```python
for k in range(K):  # Iterar sobre K capas
    for v in grafo.nodos:  # Para cada nodo
        # Agregación: combinar información de los vecinos
        a_v = AGGREGATE({h_u for u in vecinos(v)})
        # Actualización: actualizar la representación del nodo
        h_v = UPDATE(h_v, a_v)
# Usar las representaciones finales para la tarea deseada
output = READOUT({h_v for v in grafo.nodos})
```

Imagen 17. Código Python algoritmo para desarrollo de GNNs.

Como ejemplo tenemos la red de telecomunicaciones en su parte de subred, donde vemos que la aplicación de red neuronal permite tener los *embeddings* en los nodos tipo 1 y 2, agrupados considerablemente:

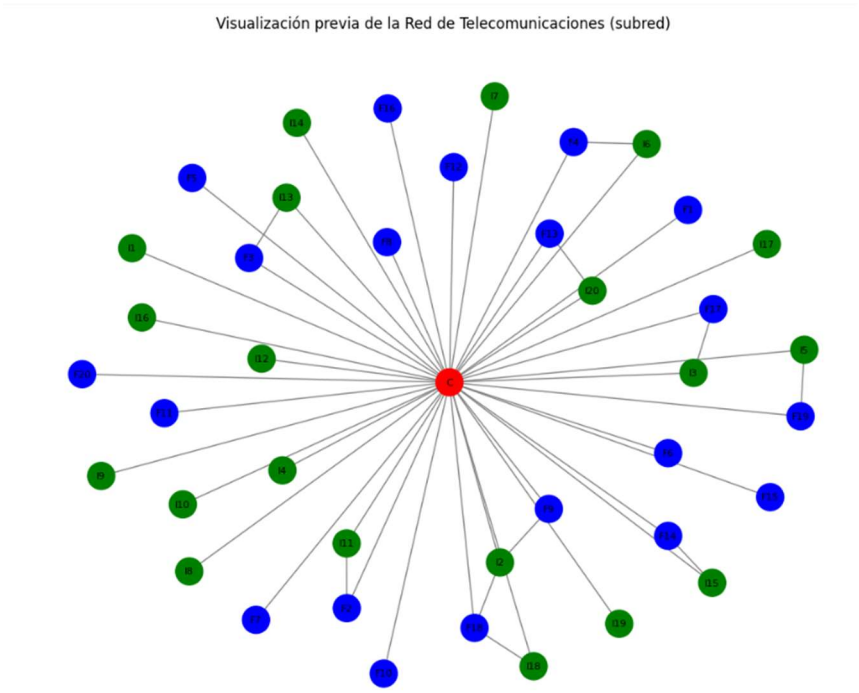
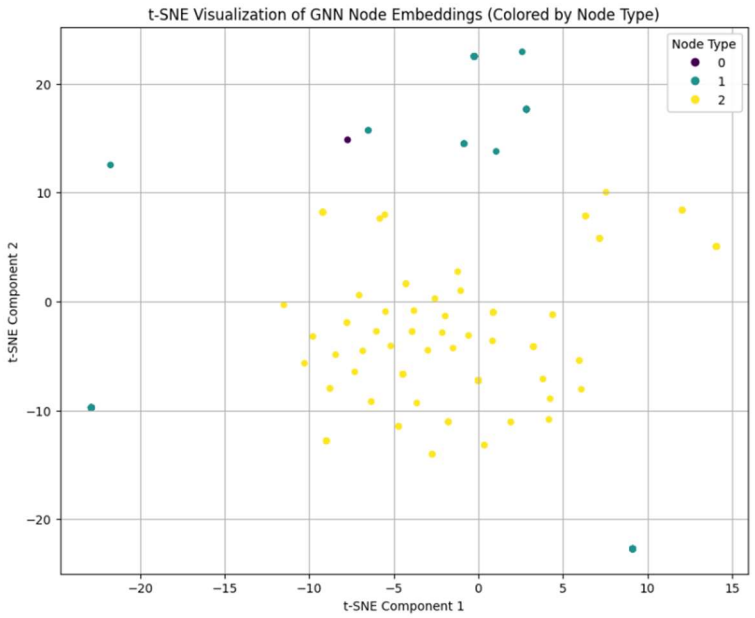


Imagen 18. Pre-visualización de una partición de la red de telecomunicaciones.




```
Interpretation of Visualization:
Nodes of the same color are nodes of the same type (Backbone, Fija, Intermitente).
If the GNN effectively captures structural information related to node types,
we would expect to see clusters of different colors in the t-SNE plot.
The degree to which these clusters are separated indicates how well the GNN
distinguishes between node types based on their position and connections in the graph.
```

Imagen 19. Visualización de GNN´s en la inserción de aprendizaje automático.

b) *Red circular y modelo de series de tiempo para predicción de carga*: En cuanto al algoritmo de series de tiempo, podemos señalar brevemente que (Ma.,y Tang. 2021). Este tipo de algoritmos combinan el análisis de datos temporales con la estructura de grafos, lo que modela y predice su evolución de sus características a lo largo del tiempo. Estos algoritmos extienden las GNNs para manejar la dimensión temporal, resultando en modelos TGNNs, (Temporal Graph Neural Networks). Por su parte, el enfoque dinámico permite agrupar una secuencia de grafos, cuyos objetivos predicen características de nodos, predicen enlaces, y clasifican grafos. De forma particular las redes recurrentes, (RNNs), son mecanismos de atención temporal que modelan la evolución. (Ma.,y Tang. 2021). Vemos en la siguiente imagen una visualización circular del agrupamiento de la red de telecomunicaciones a la que se aplica ML para calcular la serie de tiempo y evitar conglomeración, perdida de red y de saturación de uso:

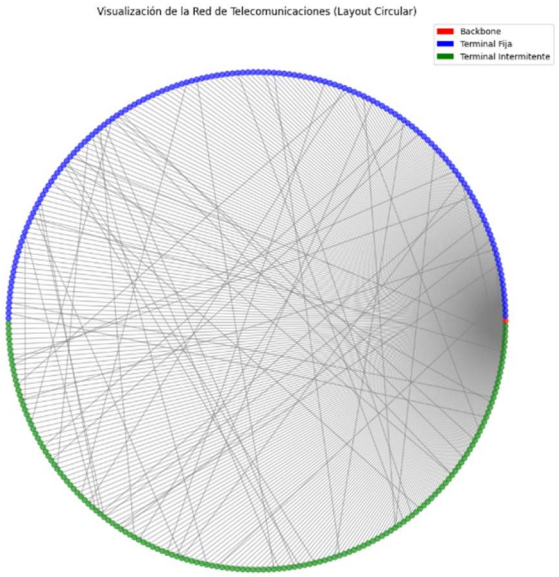


Imagen 20. Visualización de la red de telecomunicaciones en aproximación 3D al interior de la misma.

El algoritmo típico para modelar series de tiempo en grafos se basa en una extensión de las GNNs que incorpora la dimensión temporal. Este proceso puede desglosarse en los siguientes pasos: representación del grafo dinámico, con entrada, y discreción continua o discreta. Siguiendo con la propagación espacial, que es la aplicación de GNN, agregando y actualizando la información espacial para posteriormente procesar el modelo temporal, que permite enfocar en una red recurrente, (RNNs), en su atención temporal o en un modelo convolucional para capturar patrones. El algoritmo continúa con la lectura final, readout, que predice nodos, enlaces y clasifica grafos. Finalmente pasa al entrenamiento, que optimiza la función de perdida y usa retropropagación para ajustar los parámetros del algoritmo y sus componentes temporales. (Ma.,y Tang. 2021).

Existen varios modelos específicos diseñados para series de tiempo en grafos, que adaptan el marco anterior. Algunos ejemplos destacados son: Dynamic Graph Convolutional Neural Networks (DGCNN)**;** Temporal Graph Attention Networks (TGAT); EnvolveGCN, adapta grafos dinámicos con pesos de red; y TGN (Temporal Graph Networks), modelación de eventos con grafos continuos. Finalmente, DySAT (Dynamic Self-Attention Network), que permite combinar estructuras y sucesos temporales. Se observa que las ventajas son amplias, puesto que su flexibilidad permite aplicación a una gran gama de problemas reales. Tiene sus limitaciones, pues existe complejidad computacional y sobre-aislado temporal, con la carga de exceso de capas. De igual manera si los datos son escasos el modelo no es confiable. En cuanto a la implementación de estos algoritmos requieren frameworks como PyTorch Geometric o DGL en Python, como se observa en el siguiente pseudocódigo para análisis de series de tiempo: (Ma.,y Tang. 2021).

```python
for t in range(1, T):  # Iterar sobre instantes de tiempo
    # Procesar grafo en tiempo t con GNN
    for v in G_t.nodos:
        a_v = AGGREGATE({h_u for u in vecinos_t(v)})
        h_v^{(t, K)} = UPDATE(h_v^{(t, K-1)}, a_v)
    # Modelado temporal
    for v in G_t.nodos:
        h_v^{(t)} = RNN(h_v^{(t-1)}, h_v^{(t, K)})
# Predicción final
output = READOUT({h_v^{(T)} for v in G_T.nodos})
```

Imagen 21. Código Python para algoritmo ARIMA.

En el siguiente modelo de series de tiempo, vemos como el histórico de los nodos activos, en un cierto intervalo de predicción, (tiempo), muestra un descenso en la subsecución de horario de conexión de la red, como se observa en la línea de la predicción: Prophet Forecast. Como se observa en la siguiente imagen, lo que muestra que el entrenamiento se logra satisfactoriamente. Es claro el descenso en el volumen de tráfico en la red con el pasar de las horas pico y siguiendo a un equilibrio el uso de la red.

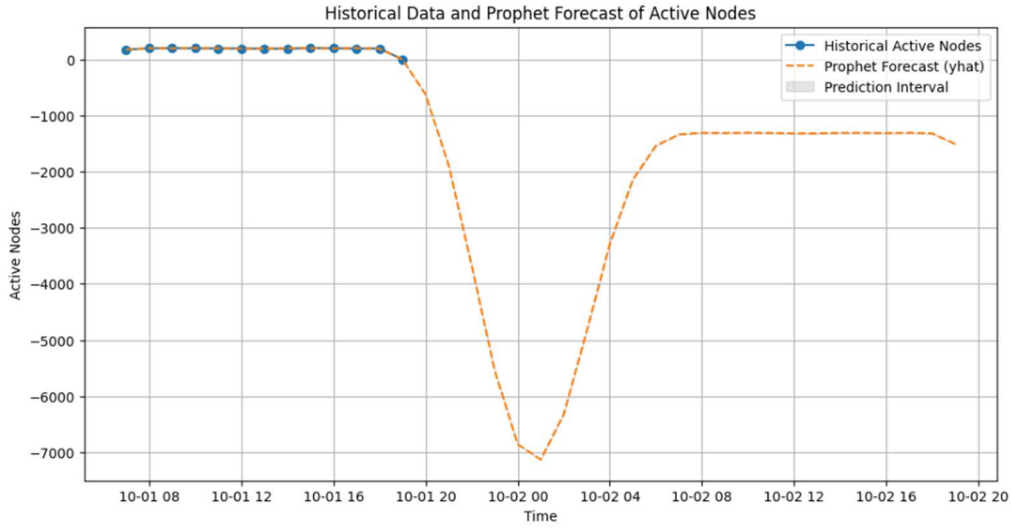


Imagen 22. Predicción de carga de ancho de banda en series de tiempo ARIMA, (Auto Regressive Integrated Moving Average).

c) *Modelo predictivo de anomalías*: la detección de anomalías es crítica en ML, tanto para ciberseguridad como mantenimiento. Una anomalía no es un simple valor atípico sino un patrón inusual en la estructura o atributos de la red, es decir, es la interacción entre un nodo y sus vecinos en el transcurso del tiempo. Los algoritmos para este trabajo contienen los siguientes elementos: definición de nodos anómalos, aristas anómalas, subgrafos y comunidades de grafos anómalas, y detección del comportamiento dinámico anómalo. (Hamilton. 2020).

En cuanto a la representación del grafo, los algoritmos de ML tradicionalmente separan vectores numéricos, por ello necesitamos incrustar los nodos o subgrafos en un espacio vectorial, (Graph Embeddings o Graph Neural Networks - GNNs). Luego entonces, los tipos de enfoque de ML para este tipo de productos pueden ser supervisados, no supervisados y semisupervisados. Para el caso particular que permite generalizar el uso de estos algoritmos de ML contamos con un caso no supervisado en la guía y consecución del aprendizaje. Siguiendo con las categorías de algoritmos de detección de anomalías, contamos con la detección de Outliers de conectividad; la detección de Outliers de Subgrafos/Comunidades. Contamos también con los atributos nodos/aristas, que son híbridos, como Outliers Atributo-Estructura; y finalmente, el enfoque de Embeddings de grafos que hemos trabajado con anterioridad como GNN´s. Como en este caso contamos con un sub-enfoque de temporalidad, estamos en un sub-enfoque que permite trabajar una serie temporal, como es este caso particular el cálculo de ARIMA, pero también contamos con los modelos de cambios de punto y recurrentes, (RGNN´s).

En el entrenamiento del algoritmo para ML, podemos observar los siguientes resultados, (Google – Colab. 2025), solamente se presentan anomalías al principio y al final de la jornada del uso de la red de telecomunicaciones, lo que implica que las anomalías se encuentran desde el comienzo o al final, pero no en el uso de la parte crítica de las conexiones predeterminadas. Esto es así debido a que la red no puede optimizarse por sí sola, sin contar con el transcurso de una cierta cantidad de conexiones predeterminadas para visualizar la cantidad de ancho de banda que podrá ser asignada a las 150 conexiones no fijas. Como se muestra en la siguiente grafica de ejes (x,y) cuadriculados entre la actividad de los nodos y el tiempo.

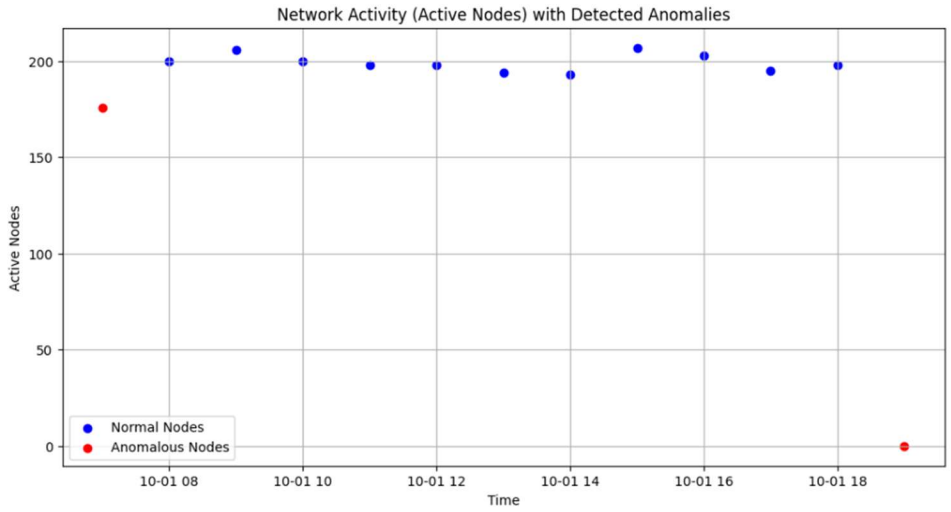


Imagen 23. Detección de anomalías en la red de telecomunicaciones.

En cuanto al código prototipo o pseudocódigo en Python, la propuesta es la siguiente para la detección de anomalías, (Ma y Tang. 2021):

python Collapse Wrap Run Copy

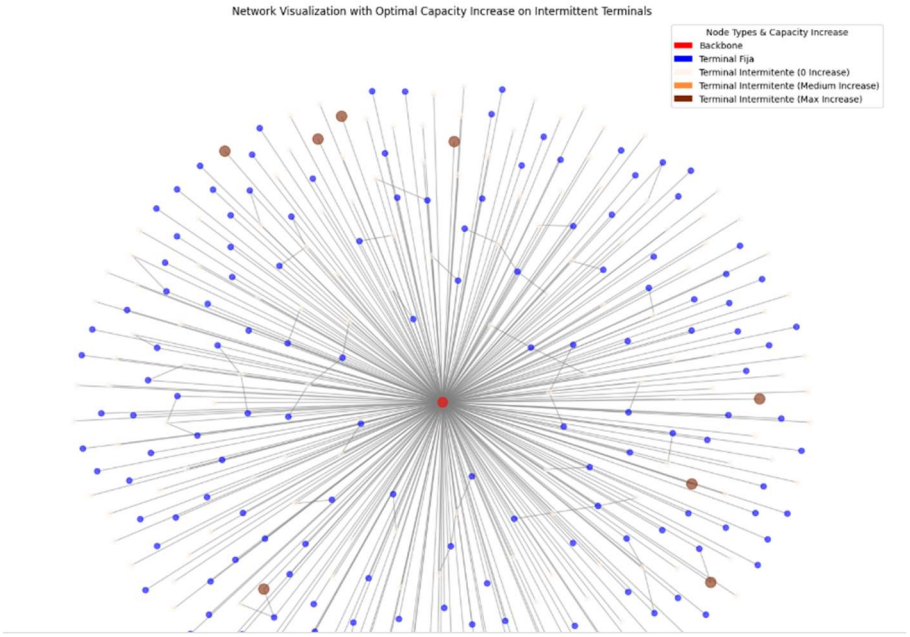
```python
# Para cada grafo (estático o en tiempo t)
for t in range(1, T):  # Para grafos dinámicos
    # Propagación espacial con GNN
    for v in G_t.nodos:
        a_v = AGGREGATE({h_u for u in vecinos_t(v)})
        h_v^{(t, K)} = UPDATE(h_v^{(t, K-1)}, a_v)
    # Modelado temporal (si aplica)
    for v in G_t.nodos:
        h_v^{(t)} = RNN(h_v^{(t-1)}, h_v^{(t, K)})
    # Calcular puntuación de anomalía
    for v in G_t.nodos:
        s_v = compute_anomaly_score(h_v^{(t)}, G_t)
# Clasificar nodos con s_v > umbral como anómalos
```

Imagen 24. Código Python para algoritmo predictivo de anomalías.

d) *Modelo de optimización de recursos:* Igualmente es de suma importancia en ML para grafos en la logística, la planificación o gestión de energía. El objetivo es asignar de manera óptima dentro de un grafo, los recursos que maximicen las métricas de rendimiento o minimización de costos. Para ello, se trata de tomar decisiones sobre la asignación del flujo de recursos. (Hamilton. 2020). Esto implica en esencia tener los datos de los recursos, los elementos del grafo, y los objetivos de optimización, vinculados a los recursos, para optimizar, minimizar, balancear o garantizar la calidad del servicio. De ahí, el aprendizaje automático interviene para optimizar esquemas que resultan ser demasiado complejos, dependientes de datos inciertos o variables, o que necesita aprender de la experiencia. El problema no se resuelve directamente sino que se potencia de varias maneras: a. Predicción de Entradas para el Optimizador Clásico. b. Aprendizaje de Heurísticas/Políticas de Optimización. c. Optimización Acelerada.

Dentro de las categorías de algoritmos ML para optimizar tenemos los principales enfoques: ML con predicción para optimización, (pre-optimización), con uso de algoritmos tipo ARIMA, prophet, random forest, y redes neuronales. Tenemos el aprendizaje por refuerzo, en donde los algoritmos RL son Q-Learning, Deep Q-Networks (DQN), Proximal Policy Optimization (PPO). Además se tiene el algoritmo GNNs que busca políticas de optimización o heurísticas que son excelentes para capturar la estructura y las interdependencias del grafo, lo que es crucial para la optimización de recursos en redes. Por ultimo tenemos la optimización combinatoria mejorada pro ML, donde el aprendizaje por búsqueda y la generación de cortes verifican la solución o acelera la resolución.

En el caso particular de la red de telecomunicaciones que se usa como modelo exploratorio y experimental, se observa que la óptima capacidad solo permite incrementar en 4 las terminales intermitentes, pero eso reduciendo la demanda de la hora pico o manteniéndola controlada a un solo ancho de banda durante todo el tiempo de la conexión ordinaria de la red.




```
Added 'capacity_increase' as a node feature. Total node features per node: 4

PyTorch Geometric Data Object:
Data(x=[301, 4], edge_index=[2, 700])

Data object details:
Number of nodes: 301
Number of edges: 700
Number of node features: 4
Has isolated nodes: False
Has self-loops: False
Is undirected: True

Shape of node features (x): torch.Size([301, 4])
Shape of edge index (edge_index): torch.Size([2, 700])
```

Imagen 25. Visualización y datos con función de optimización para la red de telecomunicaciones.

En cuanto al código prototipo o pseudocódigo en Python, la propuesta es la siguiente, para el proceso de optimización, (Ma y Tang. 2021):

python Collapse Wrap Run Copy

```
# Para cada grafo (estático o en tiempo t)
for t in range(1, T):  # Para grafos dinámicos
    # Propagación espacial con GNN
    for v in G_t.nodos:
        a_v = AGGREGATE({h_u for u in vecinos_t(v)})
        h_v^{(t, K)} = UPDATE(h_v^{(t, K-1)}, a_v)
    # Modelado temporal (si aplica)
    for v in G_t.nodos:
        h_v^{(t)} = RNN(h_v^{(t-1)}, h_v^{(t, K)})
    # Predicción de asignación
    for v in G_t.nodos:
        r_v = predict_resource(h_v^{(t)})
# Optimizar función objetivo
loss = compute_cost({r_v}, G_t)
optimize(loss)
```

Imagen 26. Código Python para código de algoritmo de optimización.

4. DISCUSIÓN DE LOS RESULTADOS.

Los grafos dinámicos y la inteligencia artificial se vinculan mutuamente por la importancia del análisis de datos. De manera particular, los grafos dinámicos son modelados realistas de sistemas complejos, lo que viene a incorporarse al desarrollo de la ciencia de redes. Por su parte, la minería de grafos nos permite conocer los conocimientos ocultos de las redes, especialmente sus estructuras, lo que permite modelar datos complejos o en gran volumen, repercutiendo directamente a las decisiones y nuevos escenarios de la realidad. De esta manera sentamos la base para generalizar el conocimiento para comprender las redes sociales, la biología, las transacciones financieras, las epidemias, la detección de patrones anómalos o temporales, etc., que son fenómenos relevantes en nuestra sociedad moderna.

Se observa que el modelo de grafo dinámico para la red se puede realizar de acuerdo a la prefiguración hipotética del problema. Se observa que se puede transformar dicho grafo en código de programación Python. Se observa que es factible para la IA clásica generar modelos de aprendizaje profundo para trabajar con los datos de la minería de grafos, permitiendo simular y optimizar los datos del grafo y la red. Es posible de igual manera para Colab generar los grafos y las gráficas estadísticas, así como visualizar los modelos de IA profunda. Se observa que la hipótesis se puede comprobar con un resultado positivo y favorable. Así como la concretización de los objetivos específicos del proyecto.

En cuanto a los resultados numéricos se presenta en la siguiente tabla la agrupación de verificabilidad, evidencia empírica y prueba de robustez. En la primera columna consideramos los criterios de respuesta de cada una de las pruebas de los algoritmos, en la segunda columna señalamos la imagen que explicita dichos resultados y en la tercera columna anotamos los resultados de codificación y recodificación en términos de segundos, en donde a menor tiempo mayor robustez, como se muestra en la siguiente tabla 2.

Verificación	***Evidencia empírica***	***Robustez***
En el modelo de detección de anomalías tenemos: El modelo de detección de anomalías identificó dos anomalías en los datos de actividad de la red. Estas anomalías se producen al inicio y al final del periodo operativo simulado: *Anomalía a las 07:00 (7:00 AM): El modelo marcó este punto porque representa el inicio del periodo activo de la red. El número de nodos activos (176) y de aristas (175) es relativamente bajo en comparación con las horas pico. Este estado inicial se desvía del comportamiento promedio durante las horas operativas principales, lo que provoca que se identifique como una anomalía.* *Anomalía a las 19:00 (7:00 PM): Este punto se detectó como una anomalía porque marca el final del periodo activo de la red. El número de nodos activos (0) y de aristas (0) cae a cero, una disminución significativa con respecto a los niveles de actividad típicos a lo largo del día. El modelo considera esta desviación como anómala.* *En esta simulación, estas anomalías no indican fallos en la red, sino que reflejan el patrón esperado de inicio y fin del uso de la red. El modelo Isolation Forest identificó correctamente estos puntos porque sus valores se encuentran fuera de la*		*12 seg.*

distribución típica de la actividad de la red observada durante la mayor parte de la ventana operativa (08:00 - 18:00).		
Modelo de predicción de carga en series ARIMA: *Prophet es una librería de código abierto desarrollada por Facebook para prever series temporales. Está diseñada para pronosticar datos con fuertes efectos estacionales (diarios, semanales, anuales), tendencias a largo plazo y la influencia de festividades. Es robusta ante valores atípicos y datos faltantes. Funciona descomponiendo la serie temporal en tres componentes principales:* *Tendencia (g(t)): Modela los cambios no periódicos en la serie temporal. Prophet puede manejar cambios de tendencia automáticamente. (Crecimiento lineal o logístico).* *Estacionalidad (s(t)): Modela los cambios periódicos, como los patrones diarios, semanales o anuales. Se implementa usando series de Fourier.* *Días festivos (h(t)): Modela los eventos irregulares que tienen un impacto predecible en la serie temporal. (No se usó explícitamente aquí, pero es una capacidad).*	*El modelo general de Prophet es: $y(t) = g(t) + s(t) + h(t) + \varepsilon(t)$* *En nuestro caso, utilizamos Prophet para pronosticar la cantidad de Nodos Activos en la red.* *Resultados del Modelo Prophet: Después de entrenar el modelo Prophet con los datos históricos de actividad de nodos por hora, obtuvimos las siguientes predicciones:* *Predicciones futuras: El modelo generó pronósticos para las Nodos Activos para las próximas 24 horas, extendiéndose más allá del período histórico disponible.* *Tendencia: Con solo un día de datos, la tendencia a largo plazo es difícil de discernir, pero el modelo capturó la forma general de la actividad de los nodos a lo largo del día.* *Estacionalidad Diaria: El modelo detectó un patrón de estacionalidad diario, mostrando un aumento en la actividad durante las horas diurnas y una disminución durante la noche, lo cual es coherente con el comportamiento esperado de una red de telecomunicaciones.* *Intervalos de Predicción: El pronóstico incluye intervalos de confianza (yhat_lower y yhat_upper), que nos dan un rango de valores probables para las futuras Nodos Activos.* *Resultados: 1) Los patrones de la actividad son claros. 2) Se contiene una base para planificación. 3) Se han limitado datos escasos. 4) Se han identificado concretamente las anomalías. LA caída del ancho de banda en la hora pico es de: 53% Entre 201-251 de uso completo de actividad por nodo.*	*13 seg.*

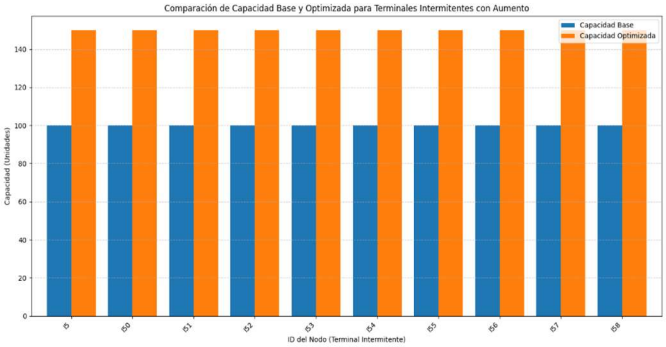
El modelo de optimización: *Muestra que la red se ha optimizado en un 3,33 % con respecto a la capacidad base. A continuación, se detallan los resultados:* *Capacidad base total (todos los nodos intermitentes): 15 000 unidades* *Aumento óptimo de la capacidad total: 500 unidades* *Porcentaje de optimización (aumento con respecto a la capacidad base): 3,33 %*		*15 seg.*

Tabla 2. Concentrado de resultados de los hallazgos principales.

5. CONCLUSIONES

La implementación del software de visualización y creación de grafos como son las librerías Gephi y otras, (Cytoscape, Graphviz, etc), son de gran utilidad y permiten generar el grafo dinámico, así como observar su temporalidad en un lapso de tiempo para conocer su comportamiento. (Gephi. 2025). Estas herramientas permiten el manejo de BD para grafos en su forma de datos, lo que contribuye al manejo de archivos de una forma versátil, gratuita y práctica. (Cytoscape. 2025).

Por su parte, el aprendizaje automático de la red tipo GNNs es factible y generalizable a un número grande de la red, del orden del 1 x 10^3. Se observa que GML es posible de generar un aprendizaje con los algoritmos en Python para introducir en lo profundo de la web IA y generar de manera inmersiva la posibilidad de visualizar

una estadística de grafos que permita la optimización de una red de telecomunicaciones, pronosticar sus anomalías y conocer sus series de tiempo. Es importante resaltar que la IA logra la transferencia de GNNs, (las redes neuronales graficas), a través de la web de manera muy específica, dependiendo del tipo de grafo y del tipo de incertidumbre para sus métricas. Esto corrobora la hipótesis que pasa de un caso de estudio a una hipótesis de factibilidad y posibilidad de desarrollo de software, manejo de código y generación de nuevo código, específicamente para Python. (Bonaccorso.2018).

En resumen, el ML no reemplaza la optimización en grafos, sino que la hace más inteligente, adaptable y escalable, permitiendo manejar la complejidad y la incertidumbre del mundo real de manera mucho más efectiva. En cuanto a los resultados de la aplicación de ML hacia una estructura de grafos como la analizada, puede generalizar la aplicación de los algoritmos de optimización, series de tiempo, búsqueda de anomalías, y análisis de redes neuronales, tanto estadístico como en su base de datos de grafos. (Barbási. 2012). Es claro a su vez, que los algoritmos de modelos predictivos, contribuye a generalizar la capacidad del aprendizaje automático a la ingeniaría. Debido a que el sistema digital inteligente, mantiene el equilibrio de carga, uso y energía de las terminales conectadas. Lo cual permite conocer los límites de las redes, como la misma ciencia de redes lo hace ver. Ya que son dependientes del ancho de banda y de la energía finita que mantiene el sistema. En este caso particular vemos el incremento de nodos en 4, que mantendría la red en un funcionamiento continuo y constante. (Barbási. 2012). Se observa también una extrapolación en los valores fuera de los rangos-horarios pico, con base en las tendencias observadas dentro de las horas de trabajo naturales. Estos son los únicos anómalos, lo que permite a la red mantener la optimización de sus recursos. Esto lo viene a corroborar el algoritmo de ML para ARIMA, serie de tiempo.

En cuanto al cómputo cuántico, podemos considerar que el desarrollo de software permite que la IA sea más rápida en sus cálculos. En este sentido, el cómputo cuántico viene a corroborar los algoritmos, agregando procesos de estimación de error, haciendo más precisos los pronósticos. Por otra parte, al igual que la IA actual, el cómputo cuántico permite visualizar el proceso del grafo dinámico en 3D, lo que sus gráficas y visualización contribuyen al entendimiento intuitivo de los procesos y algoritmos para grafos. (Yanofsky. 2008).

Por último la formulación de un esquema matemático que puede traducirse en código como es la red neuronal, GNN, (Barbási. 2012). Que en este caso se logran con el uso de las librearías de Python como son: TensorFlow, PyTorch, Keras, Scikit-learn. Estas redes permiten manejar grandes volúmenes de datos y trabajar a manera análoga al cerebro, con funciones de activación y/o perdida, que permiten modelar relaciones no lineales y patrones complejos. Van agrupando capas de código que sirven al ML y a la IA para trabajar activamente en problemas de ingeniería y sociedad. (Benítez. 2014).

Limitaciones y futuras líneas de investigación: En este caso se observa que una red más compleja permitirá analizar la capacidad de incremento de carga y la capacidad de optimización mediante una mayor cantidad de atenuaciones o limitantes de trabajo. Estas conexiones pueden ser de terminales u otra cantidad de dispositivos de la empresa o industria en cuestión. Esto permitiría re-evaluar los códigos para IA clásica y observar su comportamiento con una mayor cantidad de datos, mayor cantidad de volumen de trabajo en horas pico, y mayor cantidad de tiempo laboral, (días, semanas, meses, etc.). De igual manera para que las anomalías sean de mayor impacto se requiere considerar una serie de tiempo mayor, en días o semanas, para nuestro caso concreto, las anomalías son mínimas y su impacto no es trascendental para el uso de la red. Se espera que futuro el cómputo cuántico pueda considerar la solución a distintitas anomalías, así como la solución de optimización ante distintos

escenarios que puedan suscitarse como problemas intermedios o intermitentes en la conexión de la red, transformando esta visualización 3D en un grafo dinámico.

6. REFERENCIAS BIBLIOGRAFICAS

Barabási, A.-L. (2012). *Network science*. Cambridge University Press.

Benítez, R., Cancerrado, A., Escudero, G., & Kanaan, S. (2014). *Inteligencia artificial avanzada*. UOC.

Bonaccorso, G., Fandango, A., & Shanmugamani, R. (2018). *Python: Advanced guide to artificial intelligence. Expert machine learning systems and intelligent agents using Python*. Packt Publishing.

Bratanic, T. (2024). *Graph algorithms for data science*. Manning.

Caicedo, A., et al. (2010). *Introducción a la teoría de grafos*. Elizcom.

Chollet, F. (2018). *Deep learning with Python*. Manning Publications Co.

Cytoscape. (2025). *Network data integration, analysis, and visualization in a box*. Recuperado de https://cytoscape.org/

Even, S. (2012). *Graph algorithms*. Cambridge University Press.

Gephi. (2025). *The open graph viz platform*. Recuperado de https://gephi.org/

Google Colab. (2025). *Optimización de una red de telecomunicaciones empresarial*. Recuperado de https://colab.research.google.com/drive/1Y1Jtjek8IyDIYWPFemrmfUWIBOCPYTuU

Han, J., Kamber, M., & Pei, J. (2012). *Data mining: Concepts and techniques* (3rd ed.). Morgan Kaufmann.

Hamilton, W. (2020). *Graph representation learning*. Morgan & Claypool Publishers.

Hernández, R., & Fernández, C. (2014). *Metodología de la investigación* (6ª ed.). McGraw Hill.

IBM. (2025). *Quantum*. Recuperado de https://www.ibm.com/quantum

IBM. (2025). *IBM Quantum Platform.* Recuperado de https://quantum.cloud.ibm.com/

Iqbal, A. (2010). *Graph theory and algorithms*. Information Technology University.

Ma, Y., & Tang, J. (2021). *Deep learning on graphs*. Cambridge University Press.

Ponce, P. (2010). *Inteligencia artificial con aplicaciones a la ingeniería*. Alfaomega.

Russell, S., & Norvig, P. (2004). *Inteligencia artificial: Un enfoque moderno* (2ª ed.). Pearson Prentice Hall.

Soria, E., et al. (2022). *Inteligencia artificial: Casos prácticos con aprendizaje profundo*. RAMA Ediciones.

Witten, I. H., Frank, E., & Hall, M. A. (2011). *Data mining: Practical machine learning tools and techniques* (3rd ed.). Morgan Kaufmann.

Wu, L., Cui, P., Pei, J., & Zhao, L. (2022). *Graph neural networks: Foundations, frontiers, and applications.* Springer.

Yanofsky, N., & Mannucci, M. (2008). *Quantum computing for computer scientists*. Cambridge University Press.