
MÁSTER EN INGENIERÍA DE TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE MÁSTER

# Optimización de la asignación de recursos con IA generativa: Una herramienta de lenguaje natural para identificar regiones rentables y transportistas.

Autor: Carlos Sánchez Albertí

Director: Bernardo Villazán Gil

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Optimización de la asignación de recursos con IA generativa: Una herramienta de lenguaje natural para identificar regiones rentables y transportistas.

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2023/2024 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Carlos Sánchez Albertí Fecha: 16/ 07/ 2024

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Bernardo Villazán Gil Fecha: 24/08/ 2024


MÁSTER EN INGENIERÍA DE TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE MÁSTER

# Optimización de la asignación de recursos con IA generativa: Una herramienta de lenguaje natural para identificar regiones rentables y transportistas.

Autor: Carlos Sánchez Albertí

Director: Bernardo Villazán Gil

Madrid

**OPTIMIZACIÓN DE LA ASIGNACIÓN DE RECURSOS CON IA GENERATIVA: UNA HERRAMIENTA DE LENGUAJE**
**NATURAL PARA IDENTIFICAR REGIONES RENTABLES Y TRANSPORTISTAS.**
**Autor:** Sánchez Albertí, Carlos
**Director:** Villazán Gil, Bernardo
**Entidad Colaboradora:** ICAI – Universidad Pontificia Comillas

# RESUMEN DEL PROYECTO

El objetivo de este proyecto es crear un modelo capaz de encontrar la ruta óptima de manera rápida y eficaz. Además, se le añade a la dificultad de implementar una herramienta de Inteligencia Artificial que permita interpretar el lenguaje natural para actualizar el estado de la red.

**Palabras clave**: LLM, Supply Chain, Cross Entropy Model, CE.

## 1. Introducción

En la actualidad, la integración de la inteligencia artificial en las cadenas de suministro ofrece una oportunidad única para optimizar las rutas de transporte, reduciendo costos y mitigando el impacto ambiental. Este proyecto busca precisamente eso: una sinergia entre tecnología avanzada y logística para enfrentar los desafíos actuales. Factores como la guerra entre Rusia y Ucrania, así como los recientes conflictos en el Mar Rojo, han desestabilizado las rutas de abastecimiento, generando costos elevados que impactan tanto a las empresas como a los consumidores. Además, la variabilidad de los precios del combustible, los peajes, y la capacidad de almacenamiento en los centros logísticos son elementos que complican la planificación eficiente de las rutas. [1]

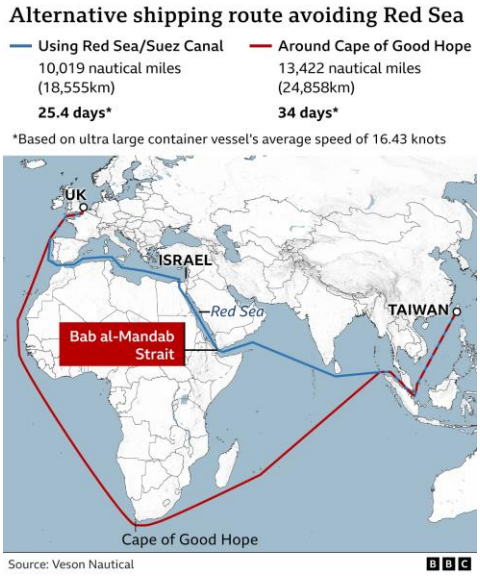



*Figura 1. Cambio producido en la ruta debido al conflicto en el mar rojo [1]*

Para abordar estos desafíos, se ha diseñado una herramienta innovadora que calculará de manera dinámica las rutas óptimas, considerando una amplia gama de factores. Esta solución está orientada a mejorar la eficiencia entre los grandes centros logísticos y las estaciones de entrega cercanas a las ciudades. Uno de los aspectos más destacados de

esta herramienta es su interfaz de usuario, que permite a los transportistas comunicarse en lenguaje natural, resolviendo problemas y ajustando rutas en tiempo real, lo que facilita una gestión más ágil y adaptativa.

La motivación detrás de este proyecto es clara: simplificar y automatizar la gestión de rutas de transporte, un proceso que hoy en día resulta costoso tanto en términos económicos como de recursos humanos. Al implementar un algoritmo que encuentre las rutas óptimas con mayor flexibilidad, las empresas podrán ahorrar significativamente, lo que a su vez beneficiará a los consumidores al reducir el costo de los productos. Además, la posibilidad de tomar decisiones en tiempo real gracias a una interfaz intuitiva representa un avance importante en la manera en que se gestionan las cadenas de suministro.

Al analizar el panorama actual de la tecnología en la cadena de suministro, encontramos que existen plataformas avanzadas como AWS Supply Chain, Microsoft Supply Chain Platform y Google Cloud Vertex AI, que ya utilizan inteligencia artificial para mejorar la eficiencia. [2] Sin embargo, este proyecto propone una solución que va un paso más allá, no solo respondiendo preguntas, sino también optimizando las rutas activamente basándose en datos en tiempo real. Este enfoque innovador no solo promete importantes ahorros, sino que también asegura una mayor resiliencia en sectores críticos como la producción de semiconductores y baterías, áreas prioritarias tanto para Estados Unidos como para Europa.

## 2. Definición del proyecto

Este proyecto se enfoca en desarrollar una herramienta avanzada de planificación y gestión de la cadena de suministro, utilizando inteligencia artificial generativa y técnicas de machine learning. La novedad principal radica en la creación de una interfaz que permita a los usuarios, como mánagers e ingenieros de los almacenes, interactuar con el sistema en lenguaje natural. Esta interfaz permitirá reportar obstáculos en tiempo real, con la capacidad de que el sistema identifique la gravedad del problema e integre esta información en la optimización de las rutas. Esta característica es innovadora, ya que, aunque empresas como Amazon están explorando el uso de inteligencia artificial para consultas sobre el estado del sistema, ninguna ha implementado aún una solución que gestione activamente problemas en las rutas.

## 3. Objetivos del proyecto

El proyecto tiene tres objetivos clave:

- Primero, desarrollar un modelo basado en un algoritmo de machine learning para optimizar las rutas en tiempo real. Este modelo tiene que ser rápido y encontrar una solución lo suficientemente buena.
- Segundo, implementar una herramienta de lenguaje natural que facilite la interacción con los usuarios para reconocer y responder a las condiciones actuales de la red. Esto permitirá a los usuarios modificar los parámetros del modelo anterior sin necesidad de programar, o realizar cambios directamente en el modelo.

- Tercero, definir un sistema de suministro artificial que permita simular pedidos y simule la actuación del sistema en tiempo discreto, para así evaluar el desempeño de la herramienta mediante simulaciones exhaustivas.

## 4. Planteamiento del problema

Antes de comenzar a desarrollar el modelo, se tiene que definir el alcance del problema. Básicamente este proyecto se centra en la simulación de la optimización de la cadena de suministro de una gran compañía especializada en productos lácteos, incluyendo leche, queso, yogur y nata. Dado que se trata de productos perecederos, es fundamental que el sistema de suministro sea rápido y eficiente para minimizar los costos y evitar la pérdida de productos.

La compañía cuenta con una red logística compuesta por cuatro grandes almacenes, conocidos como Fulfillment Centers, ubicados en diferentes regiones (Norte, Sur, Este y Oeste) alrededor de la Comunidad de Madrid, en la Figura 2 son los logos rojos. Desde estos almacenes, los productos se envían a siete Centros de Distribución (Distribution Centers) situados en las afueras de Madrid, dibujados en azul, que luego distribuyen los productos a los puntos de venta finales, denominados End-Points, que están dibujados en verde, y de allí se envían a los clientes.

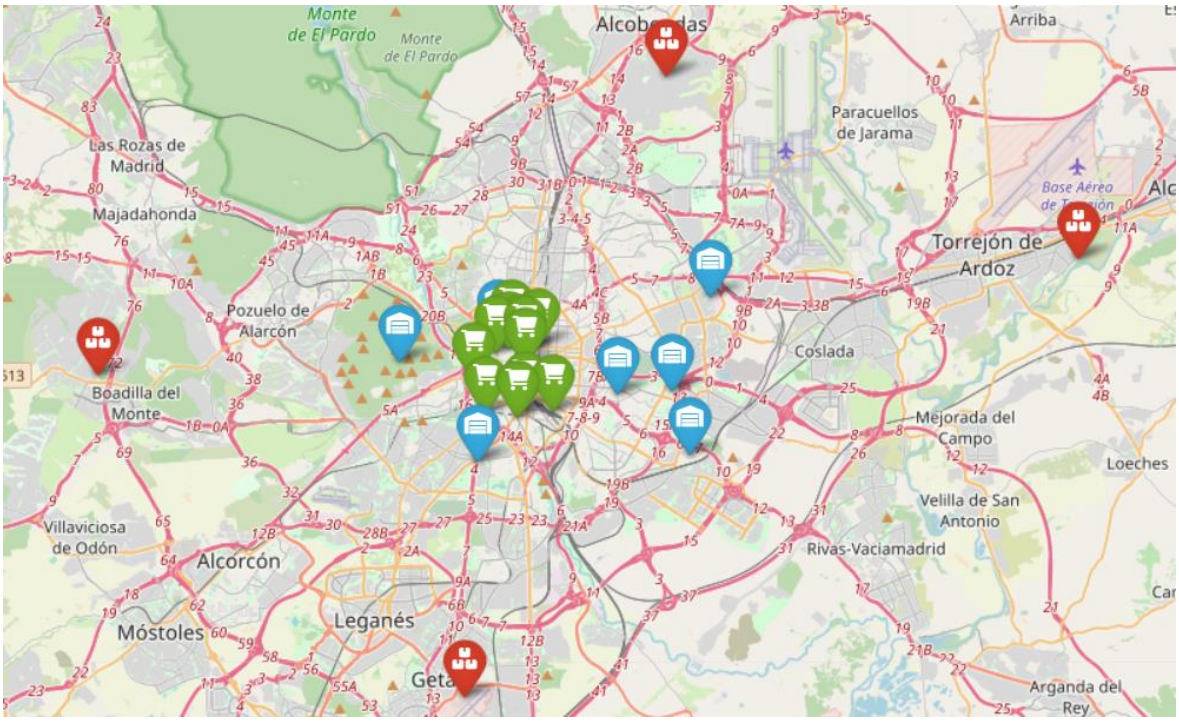



*Figura 2. Localización de los distintos almacenes*

Una vez se conoce la configuración de la red de suministro se tienen que definir cuáles serán los factores económicos que afecten al sistema. Para llevar a cabo la optimización, el proyecto tiene en cuenta varios factores cruciales. Entre ellos, la distancia entre almacenes es fundamental, ya que influye directamente en el tiempo de transporte y en los costos de combustible. La disponibilidad de productos en los Fulfillment Centers es

otro factor esencial, ya que la falta de un producto en un centro impide su transporte. Además, se consideran el tiempo de procesamiento y los costos asociados en los Centros de Distribución, donde los centros robotizados, aunque más eficientes, tienen una capacidad limitada.

La metodología para abordar este proyecto se basa en la simplificación de un sistema logístico real, permitiendo un número manejable de combinaciones para entrenar un modelo de optimización. Se han identificado y definido los factores más relevantes para el cálculo de la ruta más rentable, son los mencionados en el párrafo anterior. Estos elementos se integrarán en un algoritmo diseñado para minimizar los costos y mejorar la eficiencia de la cadena de suministro de esta empresa láctea, así como para cumplir aquellas restricciones que establece el sistema, como pueden ser el número mínimo de unidades por pedido, la capacidad de procesamiento de almacenes, o la disponibilidad del producto en los Fulfillment Centers.

## 5. Descripción del modelo

Una vez definidos claramente las partes del proyecto se procede a describir como se desarrollaron cada una.

- **Modelo de ML:** El modelo de ML seleccionado fue un modelo de Cross-Entropy (CE) debido a que fue el que mejor resultados dio en la implementación de varios modelos en un problema real[3].
  El modelo de optimización con el algoritmo de Cross-Entropy (CE) funciona generando combinaciones aleatorias de centros de cumplimiento, estaciones de distribución y puntos finales, y luego calcula el costo total de cada combinación. En cada iteración, selecciona las combinaciones más eficientes, llamadas muestras élite, y ajusta las probabilidades para generar mejores combinaciones en la siguiente ronda. Este proceso se repite hasta que se encuentra la configuración que minimiza el costo total de entrega. El resultado es una solución óptima para reducir los costos en la cadena de suministro.
  En este modelo se buscan minimizar los gastos tanto de procesamiento como el tiempo en el que el producto es procesado, minimizando la siguiente ecuación.

$$Costes = C_1 + \alpha * \beta_i * C_2$$

  Donde $C_1$es la componente de costes de procesamiento y costes de envío, mientras que la $C_2$ penaliza el tiempo de envió, siendo $\alpha$ una constante en Euros/hora para representar el coste del tiempo en el modelo y $\beta_i$ viene determinada por el modelo LLM indicando el estado de la ruta que se propone. Su valor será infinito en caso de avería.

- **Modelo LLM:** El modelo LLM utilizado para está basado en BERT para predecir la severidad de incidentes en almacenes. El modelo BERT preentrenado se ajusta para clasificar la severidad de los incidentes. Esto se hace mediante la introducción de un archivo con numerosos ejemplos de como

se busca procesar el lenguaje, es decir, un archivo con unos inputs y sus outputs esperados.
Este modelo, luego se ve implementado en el modelo anterior mediante la constante $\beta_i$.

- **Simulación:** Finalmente, se realiza un código de simulación donde se consigue generar una serie de pedidos a lo largo de un tiempo determinado y se observa el comportamiento de la red a medida que entran los pedidos.

## 6. Resultados

A continuación, se muestran los resultados del trabajo:

Para empezar, se simulará un caso donde el Fulfillment Center llamado FULL_NORTE sea el más barato con diferencia, y por lo tanto vaya mayoritariamente por allí. En la imagen se muestran los resultados de la simulación.

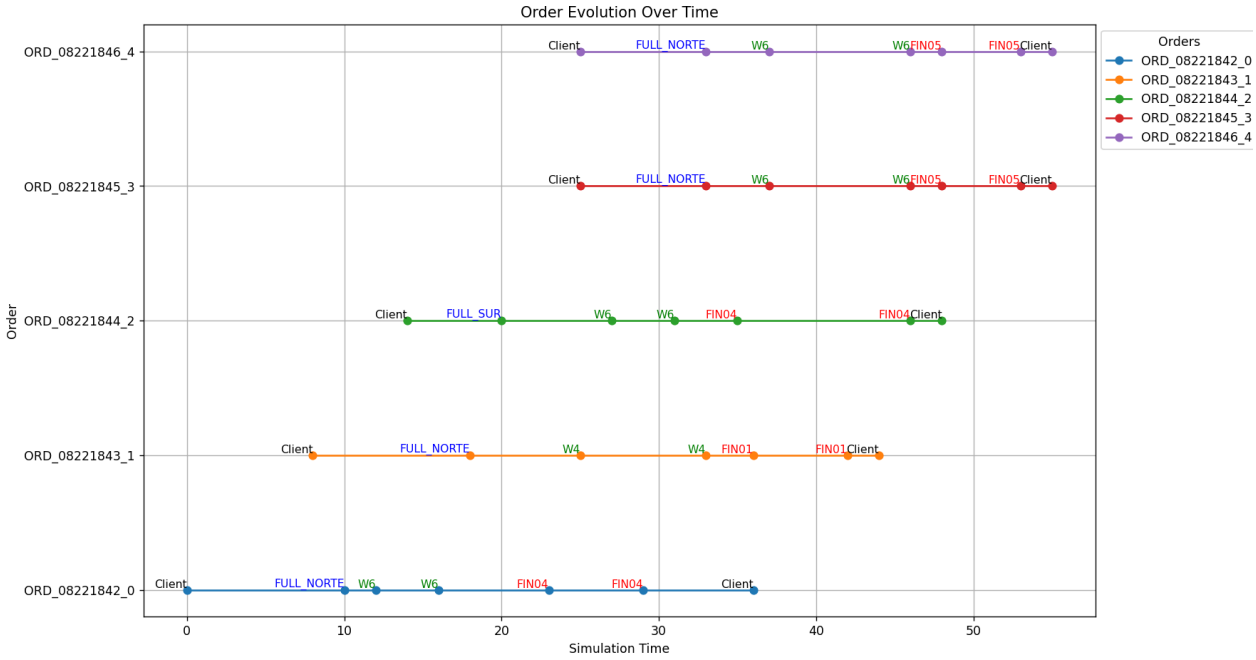



*Figura 3. Resultados de la simulación de 5 pedidos*

El siguiente paso es introducir en el modelo LLM que el almacén ya no está disponible.

```
Introduce your problem:FULL_NORTE is no longer available
The severity score for 'FULL_NORTE is no longer available' is Bad, and the entity is FULL_NORTE
The availability of warehouse FULL_NORTE has been modified to 3
```

*Figura 4. Introducción de la reestricción de FULL_NORTE en el modelo LLM*

Como se ve cambia la disponibilidad a 3 indicando que ese almacén no se puede usar más, 1 indicaría que está en completo funcionamiento y 2 en funcionamiento parcial y mejor evitar.

Se vuelve a ejecutar la simulación y se obtienen los siguientes resultados.

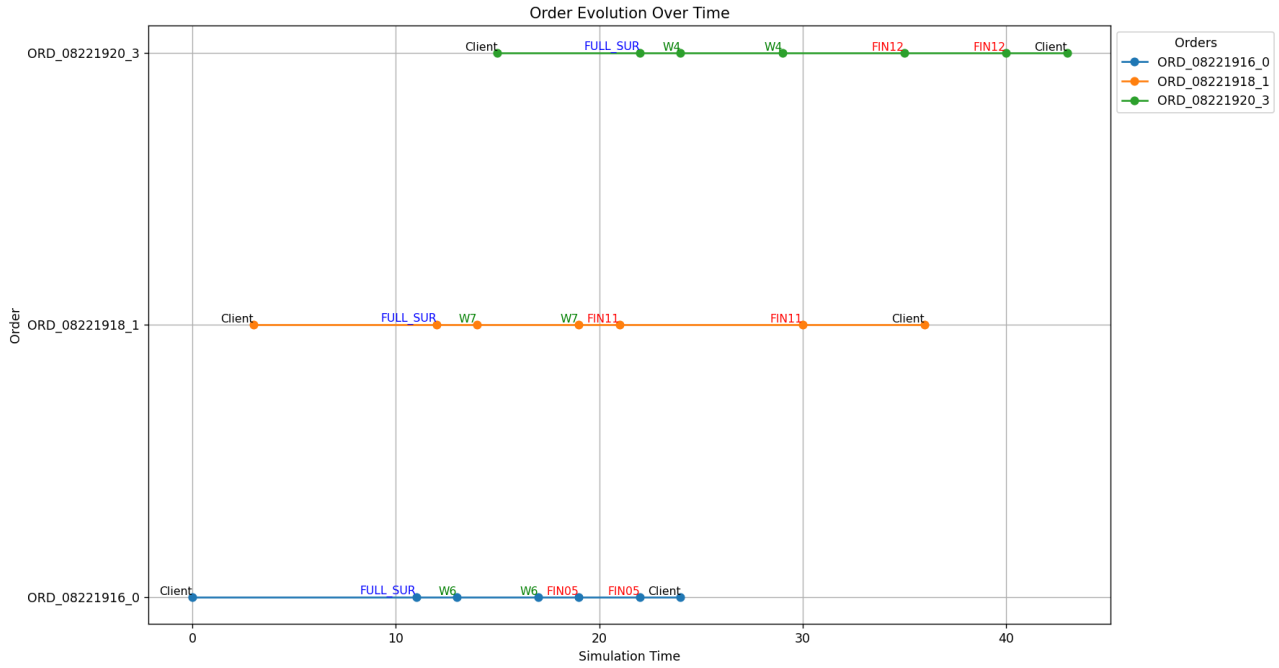



*Figura 5. Resultados tras la nueva restricción*

En estos resultados se ve como ya solo aparece FULL_SUR, y si ahora a este último se le indicase que está con capacidades reducidas y se volviera a ejecutar la simulación se obtendrían los siguientes resultados:

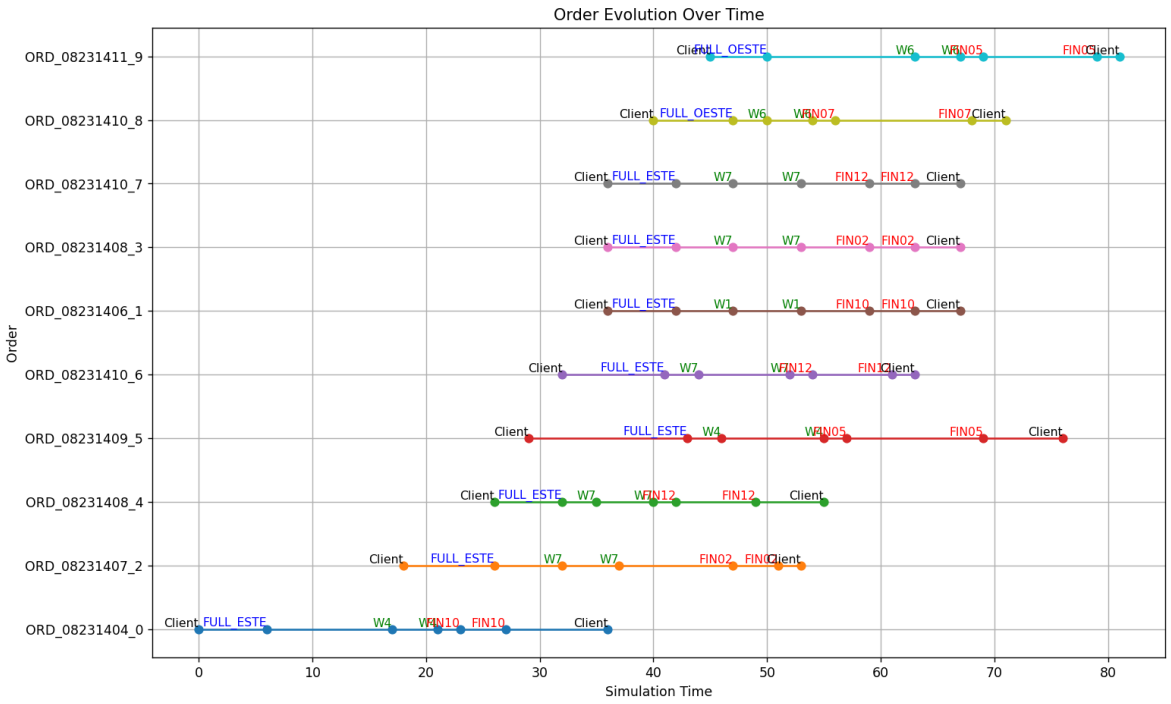



*Figura 6. Resultados con las dos reestricciones aplicadas*

Como se ve, ya no aparecen FULL_NORTE ni FULL_SUR en los resultados de la simulación. Por lo tanto, el modelo LLM, funciona correctamente.

## 7. Conclusiones

Para concluir, los resultados de la simulación son correctos y los objetivos del proyecto han sido cumplidos, ya que el modelo es capaz de encontrar una ruta subóptima, manejar todas las restricciones del sistema, como las de pedido mínimo y además, es capaz de integrar el modelo LLM en este. Además, el modelo creado para la simulación funciona correctamente permitiendo realizar grandes simulaciones y así predecir el comportamiento del modelo para una posible implementación de este en un sistema real.

Por lo tanto, este proyecto define las bases para uno más ambicioso donde se implemente el modelo mixto entre ML y IA a gran escala y en un ambiente real, permitiendo así aumentar la flexibilidad de las cadenas de suministro, así como reduciendo los costes y la contaminación de los sistemas actuales.

## 8. Referencias

[1] B. M. Race, “What do Red Sea assaults mean for global trade?,” BBC News, Dec. 19, 2023. https://www.bbc.com/news/business-67759593 (accessed May. 15, 2024).

[2] S. Ashcroft, “Top 10 AI and ML supply chain companies and solutions,” Bizclik Media Ltd, May 24, 2023. Accessed: May 15, 2024. [Online]. Available: https://supplychaindigital.com/digital-supply-chain/top-10-ai-and-ml-supply-chain-solutions

[3] B. Fahimnia, H. Davarzani, and A. Eshragh, “Planning of complex supply chains: A performance comparison of three meta-heuristic algorithms,” *Computers & Operations Research*, vol. 89, pp. 241–252, doi: 10.1016/j.cor.2015.10.008.

**OPTIMIZING RESOURCE ALLOCATION WITH GENERATIVE AI: A LANGUAGE TOOL**
**TO IDENTIFY PROFITABLE REGIONS AND CARRIERS**
**Author:** Sánchez Albertí, Carlos**.**
**Supervisor:** Villazán Gil, Bernardo
**Collaborating Entity:** ICAI – Universidad Pontificia Comillas

# ABSTRACT

The aim of this project is to create a model capable of finding the optimal route quickly and efficiently. In addition, it adds to the difficulty of implementing an Artificial Intelligence tool that allows natural language to be interpreted to update the status of the network.

**Key words**: LLM, Supply Chain, Cross Entropy Model, CE.

## 1. Introduction

Today, the integration of artificial intelligence into supply chains offers a unique opportunity to optimize transportation routes, reducing costs and mitigating environmental impact. This project seeks precisely that: a synergy between advanced technology and logistics to face current challenges. Factors such as the war between Russia and Ukraine, as well as the recent conflicts in the Red Sea, have destabilized supply routes, generating high costs that impact both businesses and consumers. In addition, the variability of fuel prices, tolls, and storage capacity in logistics centers are elements that complicate the efficient planning of routes. [1]

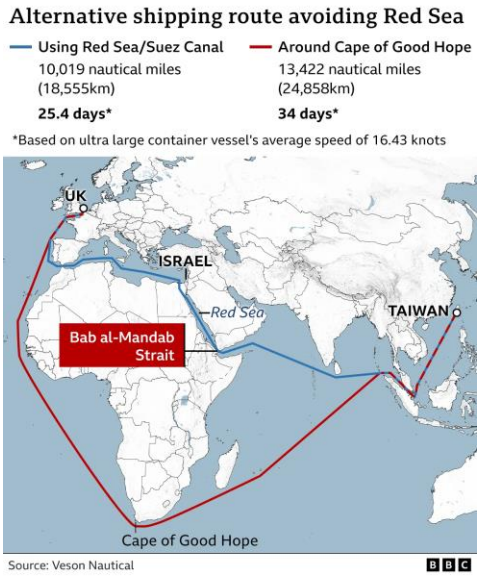



*Figure 7. Change in the route due to the conflict in the Red Sea [1]*

To address these challenges, an innovative tool has been designed that will dynamically calculate the optimal routes, considering a wide range of factors. This solution is aimed at improving efficiency between large logistics centers and delivery stations near cities. One of the highlights of this tool is its user interface, which allows carriers to

communicate in natural language, solving problems and adjusting routes in real-time, facilitating more agile and adaptive management.

The motivation behind this project is clear: to simplify and automate the management of transport routes, a process that today is costly both in economic terms and in terms of human resources. By implementing an algorithm that finds the optimal routes with greater flexibility, businesses will be able to save significantly, which in turn will benefit consumers by reducing the cost of products. In addition, the ability to make real-time decisions thanks to an intuitive interface represents an important advance in the way supply chains are managed.

When analyzing the current landscape of technology in the supply chain, we find that there are advanced platforms such as AWS Supply Chain, Microsoft Supply Chain Platform, and Google Cloud Vertex AI, which already use artificial intelligence to improve efficiency. [2] However, this project proposes a solution that goes a step further, not only answering questions, but also actively optimizing routes based on real-time data. This innovative approach not only promises significant savings, but also ensures greater resilience in critical sectors such as the semiconductor and battery production, priority areas for both the United States and Europe.

## 2. Project definition

This project focuses on developing an advanced supply chain planning and management tool, using generative artificial intelligence and machine learning techniques. The main novelty lies in the creation of an interface that allows users, such as warehouse managers and engineers, to interact with the system in natural language. This interface will allow obstacles to be reported in real time, with the ability for the system to identify the severity of the problem and integrate this information into the optimization of routes. This feature is groundbreaking, as while companies like Amazon are exploring using artificial intelligence for system status queries, none have yet implemented a solution that actively handles route issues.

## 3. Project objectives

The project has three key objectives:

- First, to develop a model based on a machine learning algorithm to optimize routes in real time. This model has to be fast and find a good enough solution.
- Second, implement a natural language tool that facilitates interaction with users to recognize and respond to current network conditions. This will allow users to modify the parameters of the previous model without the need to program, or make changes directly to the model.
- Third, to define an artificial supply system that allows simulating orders and simulates the performance of the system in discrete time, in order to evaluate the performance of the tool through exhaustive simulations.

## 4. Problem statement

Before developing the model, the scope of the problem must be defined. Basically, this project focuses on the simulation of the optimization of the supply chain of a large company specializing in dairy products, including milk, cheese, yogurt and cream. Since these are perishable goods, it is critical that the supply system is fast and efficient to minimize costs and prevent product loss.

The company has a logistics network made up of four large warehouses, known as Fulfillment Centers, located in different regions (North, South, East and West) around the Community of Madrid, in Figura 2 are the red logos. From these warehouses, the products are sent to seven Distribution Centers located on the outskirts of Madrid, drawn in blue, which then distribute the products to the final points of sale, called End-Points, which are drawn in green, and from there they are sent to customers.

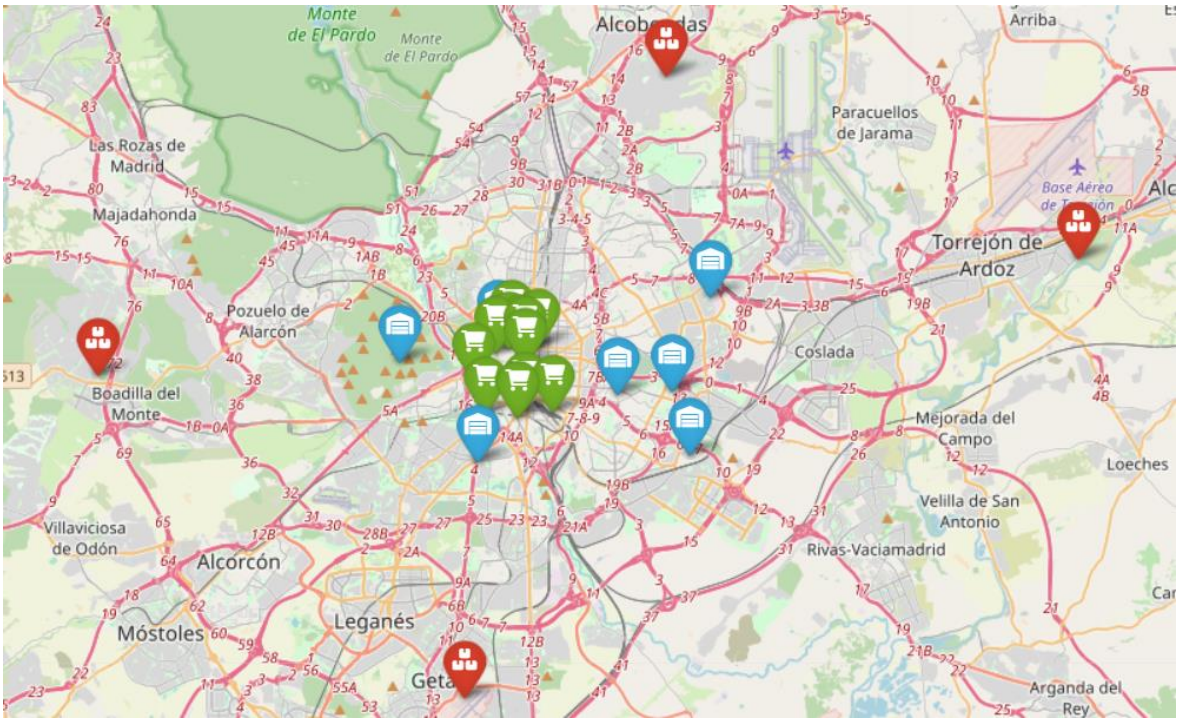



*Figure 8. Location of the different warehouses*

Once the configuration of the supply network is known, the economic factors that affect the system must be defined. To carry out the optimization, the project takes into account several crucial factors. Among them, the distance between warehouses is essential, as it directly influences transport time and fuel costs. The availability of products in Fulfillment Centers is another essential factor, since the lack of a product in a center prevents its transportation. In addition, processing time and associated costs in Distribution Centers are considered, where robotic centers, although more efficient, have limited capacity.

The methodology to tackle this project is based on the simplification of a real logistics system, allowing a manageable number of combinations to train an optimization model. The most relevant factors for the calculation of the most profitable route have been identified and defined, they are those mentioned in the previous paragraph. These

elements will be integrated into an algorithm designed to minimize costs and improve the efficiency of the supply chain of this dairy company, as well as to comply with those restrictions established by the system, such as the minimum number of units per order, the processing capacity of warehouses, or the availability of the product in the Fulfillment Centers.

## 5. Model Description

Once the parts of the project have been clearly defined, we proceed to describe how each one was developed.

- **ML Model:** The ML model selected was a Cross-Entropy (CE) model because it was the one that gave the best results in the implementation of several models in a real problem[3].

  The optimization model with the Cross-Entropy (CE) algorithm works by generating random combinations of fulfillment centers, distribution stations, and endpoints, and then calculates the total cost of each combination. In each iteration, select the most efficient combinations, called elite samples, and adjust the odds to generate better combinations in the next round. This process is repeated until the configuration that minimizes the total cost of delivery is found. The result is an optimal solution for reducing costs in the supply chain.
  In this model, the aim is to minimize both processing costs and the time in which the product is processed, minimizing the following equation.

  $$Costs = C_1 + \alpha * \beta_i * C_2$$

  Where is the component of processing costs and shipping costs, while the shipping time is penalized, being a constant in Euros/hour to represent the cost of time in the model and is determined by the LLM model indicating the status of the route that is proposed. Its value will be infinite in the event of a breakdown.$C_1 C_2 \alpha \beta_i$

- **LLM Model:** The LLM model used for is based on BERT to predict the severity of incidents in warehouses. The pre-trained BERT model is tuned to classify the severity of incidents. This is done by introducing a file with numerous examples of how language is sought to be processed, that is, a file with inputs and their expected outputs. This model is then implemented in the previous model by the constant .$\beta_i$
- **Simulation:** Finally, a simulation code is made where a series of orders are generated over a certain time and the behavior of the network is observed as the orders come in.

## 6. Results

The results of the work are shown below:

To begin with, a case will be simulated where the Fulfillment Center called FULL_NORTE is the cheapest by far, and therefore mostly goes there. The results of the simulation are shown in the image.

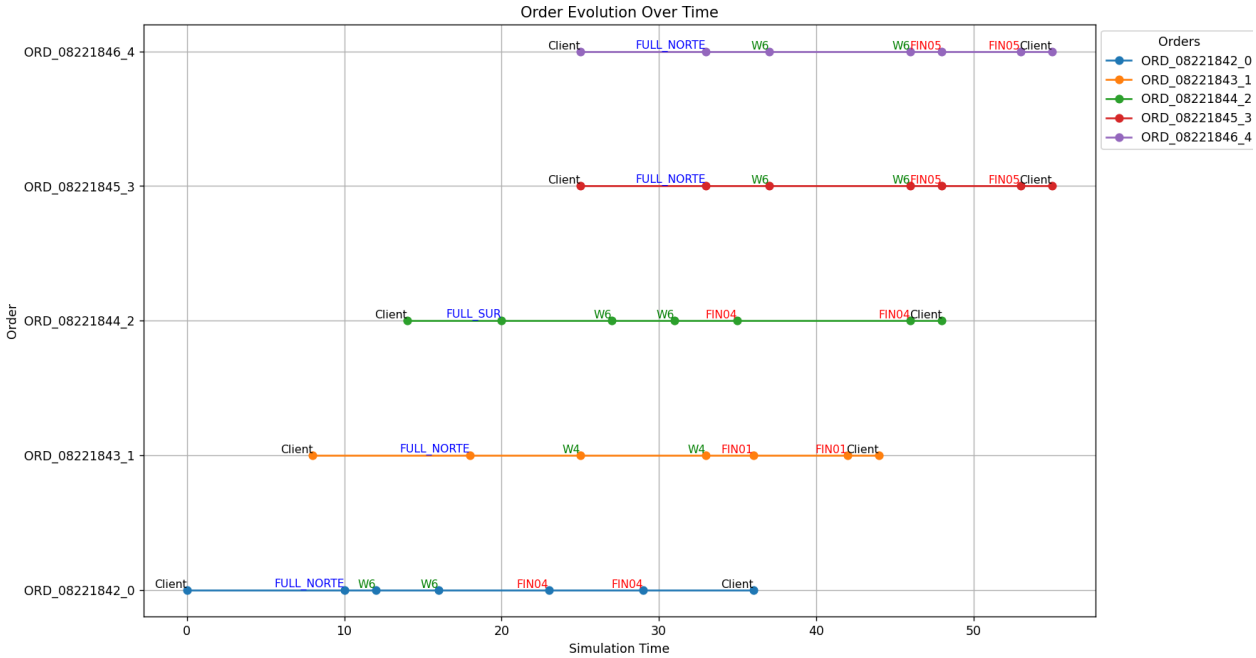



*Figure 9. Simulation results of 5 orders*

The next step is to enter into the LLM model that the warehouse is no longer available.

```
Introduce your problem:FULL_NORTE is no longer available
The severity score for 'FULL_NORTE is no longer available' is Bad, and the entity is FULL_NORTE
The availability of warehouse FULL_NORTE has been modified to 3
```

*Figure 10. Introduction of FULL_NORTE Restriction in the LLM Model*

As you can see, change the availability to 3 indicating that this warehouse can no longer be used, 1 would indicate that it is fully operational and 2 in partial operation and better avoided.

The simulation is run again and the following results are obtained.

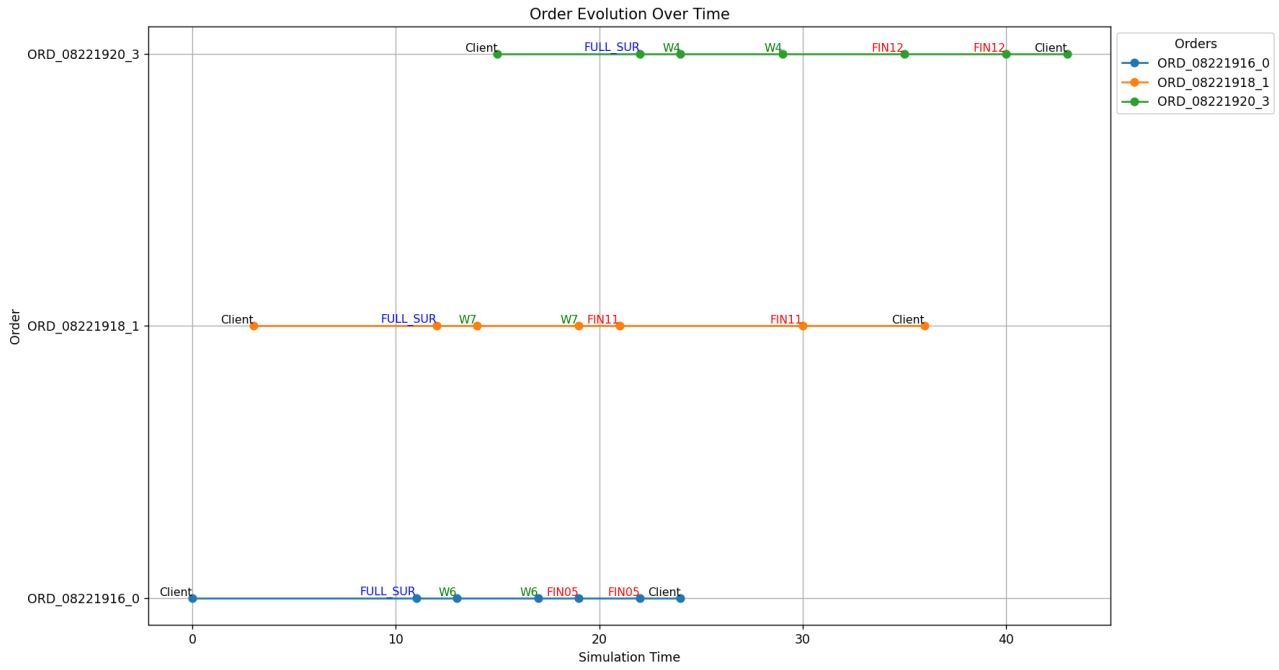



*Figure 11. Results after the new restriction*

In these results it is seen how only FULL_SUR appears, and if the latter were now indicated that it is with reduced capacities and the simulation was run again, the following results would be obtained:

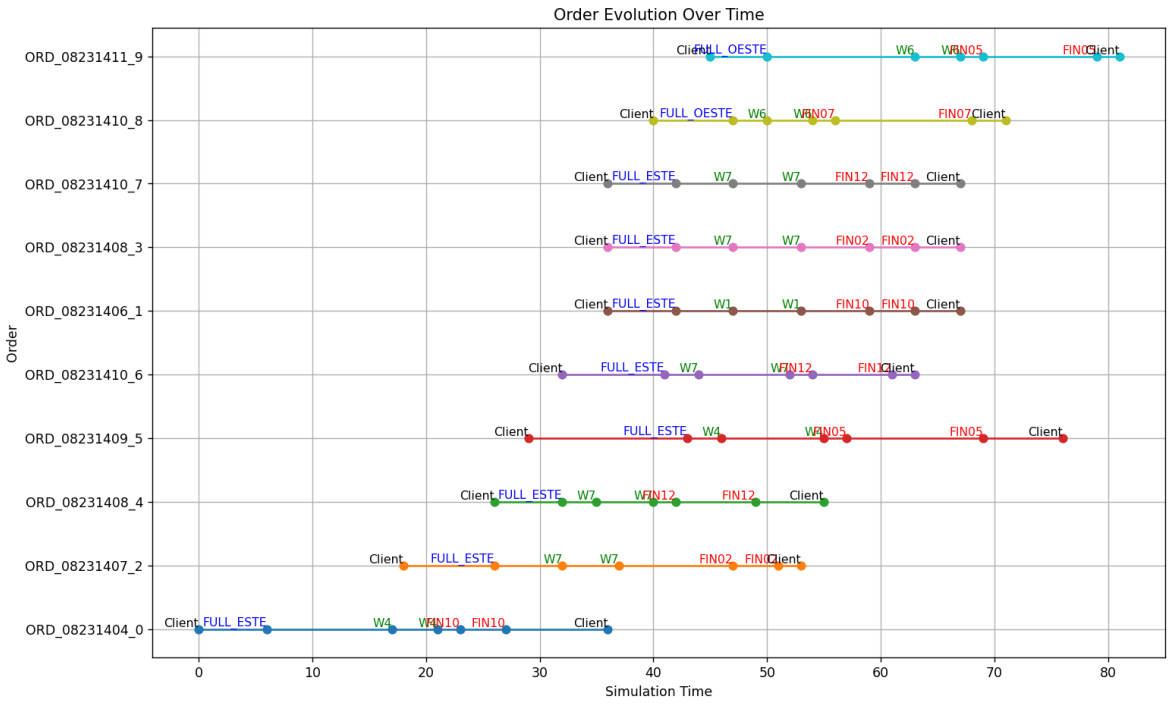



*Figure 12. Results with the two restrictions applied*

As you can see, FULL_NORTE and FULL_SUR no longer appear in the simulation results.

Therefore, the LLM model works correctly.

## 7. Conclusions

To conclude, the results of the simulation are correct and the objectives of the project have been met, since the model is able to find a suboptimal route, handle all the constraints of the system, such as those of minimum order and is also able to integrate the LLM model into it. In addition, the model created for the simulation works correctly, allowing large simulations to be carried out and thus predict the behavior of the model for a possible implementation of it in a real system.

Therefore, this project defines the basis for a more ambitious one where the mixed model between ML and AI is implemented on a large scale and in a real environment, thus increasing the flexibility of supply chains, as well as reducing the costs and pollution of current systems.

## 8. References

[4] B. M. Race, “What do Red Sea assaults mean for global trade?,” BBC News, Dec. 19, 2023. https://www.bbc.com/news/business-67759593 (accessed May. 15, 2024).

[5] S. Ashcroft, “Top 10 AI and ML supply chain companies and solutions,” Bizclik Media Ltd, May 24, 2023. Accessed: May 15, 2024. [Online]. Available: https://supplychaindigital.com/digital-supply-chain/top-10-ai-and-ml-supply-chain-solutions

[6] B. Fahimnia, H. Davarzani, and A. Eshragh, “Planning of complex supply chains: A performance comparison of three meta-heuristic algorithms,” *Computers & Operations Research*, vol. 89, pp. 241–252, doi: 10.1016/j.cor.2015.10.008.

# *Índice de la memoria*

# *Índice de figuras*

# *Índice de tablas*

# Capítulo 1. INTRODUCCIÓN

El objetivo de este proyecto es sinergia entre los mundos de la cadena de suministro y el de la inteligencia artificial, para disminuir los costes de transporte y procesamiento de los productos y su impacto en el medio ambiente.

En los siguientes apartados se realizará una introducción al tema a abordar.

## *1.1 CONTEXTO DEL PROYECTO*

En el mundo actual, las rutas de abastecimiento se encuentran en un cambio continuo donde la mayoría de las empresas que se dedican a este sector se ven obligadas a cambiar el flujo de sus camiones, barcos o trenes de mercancías debido al incremento de conflictos internacionales que están surgiendo como pueden ser la guerra de Rusia contra Ucrania, o el conflicto provocado por los ataques hutíes en el mar rojo a principios de 2024 [1] que debilitan las rutas de transporte provocando grandes cambios en las rutas, que además suponen altísimos costos para la empresa transportista y que luego repercuten en el consumidor.

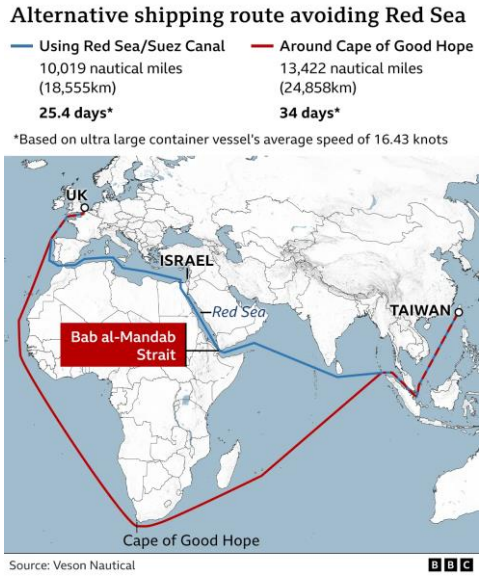



*Ilustración 1. Cambio producido en la ruta debido al conflicto en el mar rojo [2]*

Sin embargo, no solo influyen factores geopolíticos en estos cambios de rutas, sino que también influyen factores económicos, como podría ser el precio de la gasolina de una zona o país, los peajes, el número de conductores disponibles, el costo de transporte de las unidades, el costo de almacenamiento de estas en los grandes centros logísticos y muchos factores más que son dinámicos y que provocan cambios constantes en las rutas óptimas. También se tienen en cuenta factores logísticos como la rapidez de entrega o la capacidad de almacenamiento de los diversos centros logísticos, que juegan un papel crucial en evitar el colapso de la red y en satisfacer las demandas de los clientes.

Claramente, la tarea de planear, recalcular e implementar las nuevas rutas cada vez que hay un cambio en algún factor económico, geopolítico o logístico es muy tediosa y compleja. Por eso, a lo largo de los últimos años y con el gran avance que se ha producido en el campo de la inteligencia artificial generativa se ha pretendido fusionar estos dos mundos para conseguir cadenas de suministros eficientes y sobre todo flexibles ante los continuos cambios que se producen.

A raíz de lo explicado en la introducción y mediante la realización de este trabajo de fin de Máster, se busca crear una herramienta capaz de calcular de manera dinámica la ruta óptima teniendo en cuenta el mayor número de factores posible, entre ellos los mencionados

previamente. El trabajo se va a centrar en las rutas que se llevan a cabo entre los grandes centros logísticos situados en las afueras de las ciudades, donde se almacenan un gran número de productos, hasta las estaciones de entrega, que son aquellos almacenes que están ya más próximos a las ciudades y donde tienen una menor capacidad de almacenamiento pero sin embargo, se encuentran ya segmentados conteniendo solamente aquellos productos que se distribuirán en localizaciones más próximas como tiendas minoristas o centros de última milla.

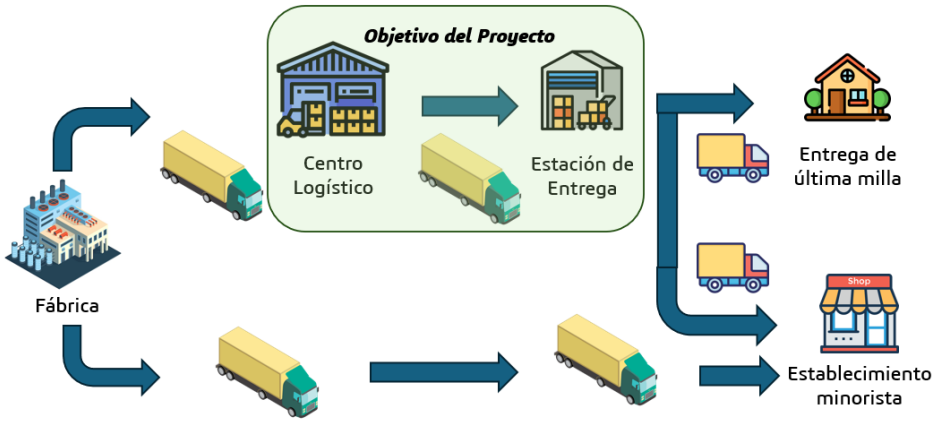



*Ilustración 2. Esquema simplificado de la cadena de suministro*

La principal característica de este trabajo es que la herramienta tendrá una interfaz de usuario con la cual los transportistas podrán escribir en lenguaje natural y el sistema les solventará las dudas y le proporcionará las nuevas rutas de manera dinámica. El motivo de esta herramienta es usar los activos de la empresa como pueden ser sus trabajadores para según sus indicaciones adaptar las rutas de una manera más dinámica y en tiempo real, mediante la interpretación del mensaje enviado por estos cada vez que tengan un problema.

## *1.2 MOTIVACIÓN*

La motivación surge para solventar el problema que sufren muchas empresas del mundo de la logística donde el hecho de la gestión, planificación y manutención de las rutas de transporte les supone un esfuerzo económico enorme, debido a la contratación de personal cualificado para realizar las tareas, así como el tiempo que estos profesionales necesitan para

establecer las rutas. Además, se busca automatizar este proceso mediante el uso de un algoritmo que sea capaz de encontrar las rutas óptimas con gran flexibilidad. Esto supondrá grandes ahorros de costes para la compañía, y también para el planeta debido a que se gastarán menos recursos.

En consecuencia, al reducir el precio del transporte impactará muy positivamente en los usuarios debido a que estos tendrán acceso a mercancías provenientes de otros mercados a un precio más asequible, por lo que crearía un beneficio importante a la sociedad.

Finalmente, para darle un paso más de innovación a este proyecto, lo que se busca es crear una interfaz que permita al usuario hablar en lenguaje natural con ella, y gracias a estos datos aportados por el usuario, el sistema será capaz de tomar decisiones en tiempo real, en función de los inputs proporcionados y un algoritmo que los clasifique.

# Capítulo 2. ESTADO DE LA CUESTIÓN

La implementación de una inteligencia artificial o de otras técnicas de decisiones automáticas como puede ser machine learning, no es una novedad en el mundo de las cadenas de suministro. De hecho, estas herramientas llevan en el mercado el tiempo suficiente como para madurar. Entre ellas destacan algunas como AWS SageMaker, la cual es la tecnología que está más ampliamente extendida y que fue lanzada en 2017. Además, sobre esta tecnología, Amazon Web Services crea su servicio de cloud AWS Supply Chain que es capaz de unificar los datos, extraer conocimiento de ellos para conectar la cadena de suministro de una empresa y los recursos que esta tiene disponible para hacer frente a la demanda. Amazon no es la única compañía que proporciona servicios en 2022 Microsoft sacó al mercado la llamada Supply Chain Platform, construida en Azure y Google tampoco se quedó atrás con su servicio llamado Google Cloud Vertex AI. [5]

A continuación, se describirán los servicios previamente mencionados más detalladamente:

## *2.1 AWS SUPPLY CHAIN*

Amazon supply chain proporciona servicios de machine learning en la nube, además sirve como tecnología de unificación de datos en la nube. Entre sus beneficios principales se encuentra la posibilidad de conectarse a la planificación de recursos empresariales (ERP) y a los sistemas de administración de la cadena de suministro existentes, sin necesidad de redefinir la plataforma.

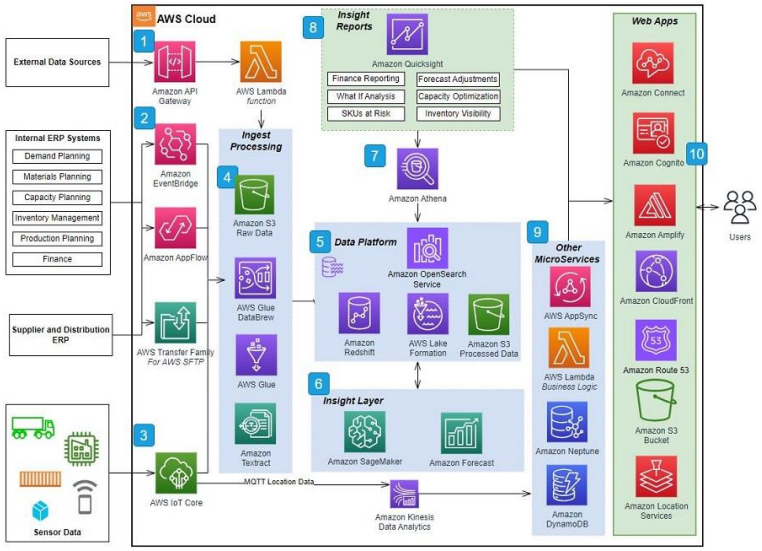



*Ilustración 3. Arquitectura de referencia de AWS Supply Chain 3 (SC3) [5]*

Cabe destacar que este servicio tiene en desarrollo una herramienta que pretende usar tecnología artficial generativa para responder a preguntas de los clientes. Todavía no está disponible este servicio, además, a diferencia de la herramienta realizada en este trabajo, solo cuenta con un servicio de preguntas acerca de como funcionaría la cadena de suministro y no usaría esos inputs para optimizar las rutas. [7]

## 2.2 MICROSOFT SUPPLY CHAIN PLATFORM

Esta plataforma utiliza la tecnología de inteligencia artificial de Microsoft para gestionar y procesar los petabytes de data que tienen las empresas, y obtener soluciones personalizadas para obtener una visión general de la cadena de suministro.

Esta plataforma tiene como componente principal el denominado Microsoft Supply Chain Center (MSCC), que permite armonizar los datos provenientes de distintos sistemas de medidas de la cadena de suministro, y mediante Data Manager, también contenido en MSCC, se procesa y se ingestan los datos. [8]

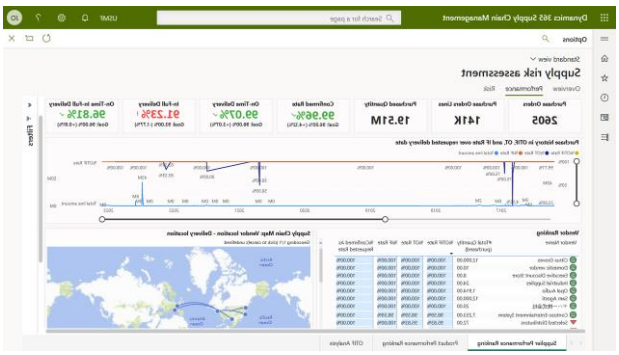


*Ilustración 4. Ejemplo de la plataforma [9]*

Finalmente, después de haber repasado las principales tecnologías existentes, se puede concluir que es un sector que todavía tiene un gran margen de desarrollo debido al enorme dinamismo que sufren las cadenas de suministro y es allí donde la inteligencia artificial puede dar un gran paso para tomar las decisiones de manera más rápida, eficiente y segura. De hecho, un estudio realizado por la consultora Mckinsey afirma que es la combinación de la IA con la cadena de suministros donde cabe la posibilidad de obtener un ahorro mayor, tal y como se puede ver en la siguiente figura[1]:

[1] M. Chui, B. Hall, H. Mayhew, A. Singla, and A. Sukharevsky, "Michael Chui," McKinsey & Company, Dec. 06, 2022. https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-in-2022-and-a-half-decade-in-review#/ (accessed May 15, 2024).

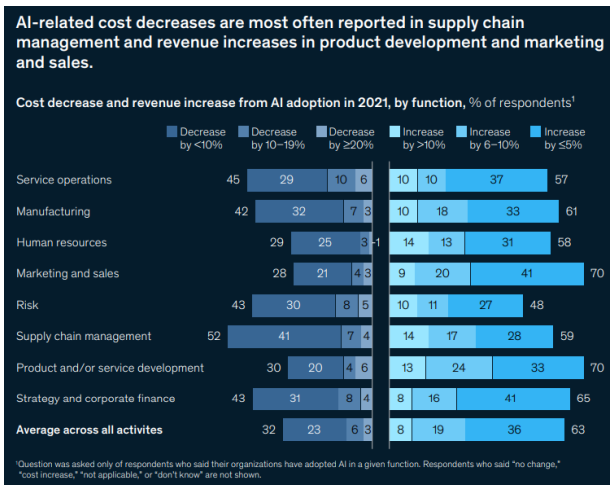



*Ilustración 5. Variación de los costes y los ingresos después de la aplicación de la IA.*

Tal y como se puede apreciar en el gráfico, vemos que la cadena de suministro tras aplicar la IA reduce un 52% sus costes y aumenta hasta en un 59% sus ingresos. Es por esta razón y por la necesidad de proteger sectores críticos como son el de los semiconductores, activos farmacéuticos o baterías de amplia capacidad que la administración de Joe Biden, el presidente de los EEUU que ha ordenado la creación de un consejo para conseguir una cadena de suministro resiliente aplicando las nuevas tecnologías de la IA. Europa ha tomado una decisión similar en este aspecto.[10]

Estas decisiones demuestran que hay un increíble potencial para desarrollar tecnologías innovadoras, y es por ello que mediante la realización de este trabajo, se busca encontrar un nuevo enfoque a partir de las tecnologías ya existentes, para mejorar los resultados.

# Capítulo 3. DEFINICIÓN DEL PROYECTO

## 3.1 JUSTIFICACIÓN

Tal y como se ha comentado anteriormente, este trabajo consistirá en el desarrollo de una herramienta de planificación y gestión de la cadena de suministro, mediante el uso de la inteligencia artificial generativa y otras técnicas avanzadas como machine learning. La principal característica de este proyecto es el desarrollo de una interfaz que permita a los usuarios del sistema, como los transportistas, los ingenieros de supply chain, o cualquier otro usuario hablar en lenguaje natural para informar de posibles obstáculos en el camino y que el sistema sea capaz de identificar la gravedad del problema y así poder gestionar ese input en la ecuación de cálculo de la ruta óptima.

Esta característica, es una gran novedad que todavía no ha sido implementada por ninguna empresa en el mercado. De hecho, la única empresa que esta cerca de realizar algo parecido es Amazon, con la futura implementación de una IA para realizarle preguntas acerca del estado del sistema. Destacar que no mencionan en ningún momento que vayan a usar esa IA para reportar posibles problemas en las rutas establecidas.

## 3.2 OBJETIVOS

Los principales objetivos del proyecto serán los cuatro siguientes:

1. Desarrollo de un modelo simple basado en ML y IA generativa que sea capaz de optimizar las rutas en tiempo real.
2. Implementación de una herramienta de lenguaje natural que permita la interacción con los usuarios para reconocer el estado actual de la red y poder tomar decisiones en función de los inputs recibidos.

3. Entrenar el modelo de IA generativa con un conjunto de datos sintéticos de gran tamaño, donde se incluirá información de regiones, transportistas, métricas de rentabilidad y otros factores que influyan en la obtención de la ruta óptima.
4. Evaluación del desempeño de la herramienta mediante simulaciones.

## *3.3 METODOLOGÍA*

Para lograr cumplir el objetivo del proyecto, se va a dividir en cuatro partes claramente diferenciadas, que son las siguientes:

1. **Creación y generación del modelo de datos:** En este apartado, se definirán que datos hacen falta para resolver este problema. Para ello se creará el modelo de datos con todas las tablas necesarias. Posteriormente, se procederá a la población de las tablas mediante el uso de datos sintéticos.
2. **Definición del modelo de Inteligencia Artificial:** Aquí se buscará encontrar el modelo que mejor se adapte a las condiciones para resolver el problema. Claramente, la primera parte será una comparativa de los distintos modelos y algoritmos existentes. Una vez elegido el tipo de modelo que se necesita, se procederá a desarrollarlo.
3. **Desarrollo de la interfaz de usuario:** El tercer paso, consistirá en el desarrollo de la interfaz de usuario, donde se creará un modelo que sea capaz de procesar los inputs de los usuarios, clasificarlos en un rango de respuesta buena o mala, que luego se usará en el modelo desarrollado en el punto anterior. Una vez más, el primer paso, será realizar un estudio de los tipos de modelos que existen actualmente y diseñar uno para conseguir el correcto funcionamiento.
4. **Entrenamiento del modelo:** Finalmente, se realizará un entrenamiento del modelo, comparando entre el resultado del modelo usando los inputs de la interfaz de usuario creada y el mismo modelo sin esta.

## *3.4 PLANIFICACIÓN Y ESTIMACIÓN ECONÓMICA*

### 3.4.1 DISTRIBUCIÓN DEL TRABAJO

En la Tabla 1 se muestra cómo se distribuirá la carga de trabajo a lo largo de los próximos meses.

*Tabla 1. Distribución del Trabajo*

| | | |
|---|---|---|
| Definición del proyecto y planteamiento del problema | 01/05/24 hasta 15/01/24 | 1. Definición de los objetivos<br>2. Primer planteamiento del sistema |
| Creación y desarrollo del modelo | 16 /05/24 hasta 31/06/24 | 1. Creación y definición del modelo de datos<br>2. Creación y desarrollo del algoritmo para resolver el camino óptimo<br>3. Creación de la interfaz |
| Entrenamiento del modelo | 01/07/24 hasta 15/07/24 | 1. Test del modelo sin la interfaz<br>2. Test completo del modelo |

### 3.4.2 RECURSOS

Para el desarrollo del trabajo se necesitarán las siguientes herramientas de software:

- La primera de todas será MySQL, la cual es una herramienta que sirve para crear modelos de datos con todas las tablas necesarias para luego almacenar los datos sintéticos. Además, está aplicación permite conectarse a la base de datos a través de Python.
- La segunda herramienta donde se trabajará es Visual Studio Code, este será el entorno de programación donde se desarrollará el código de Python.
- Por último, y tal como se ha mencionado en los puntos anteriores el lenguaje de programación será en Python.

# Capítulo 4. PLANTEAMIENTO DEL PROBLEMA

## *4.1 INTRODUCCIÓN AL PROBLEMA*

Para la realización de este proyecto se va a simular la optimización de la cadena de suministro de una gran compañía alimenticia especializada en productos lácteos. Esta compañía tiene concretamente 4 productos, que son los siguientes, leche de vaca, queso, yogurts y briks de nata. Al ser una empresa alimenticia, es clave que el sistema de suministro sea rápido y con costes bajos para así evitar que perezca el producto, así como la posible disminución de los márgenes debido a una costosa cadena de suministro.

Esta empresa tiene de una red logística que consta de 4 grandes almacenes situados en las afueras de la Comunidad de Madrid (denominados Fulfillment Centers) donde se almacenan grandes cantidades de cada producto antes de ser enviados a los centros de distribución. Estos cuatro centros están localizados en cuatro regiones distintas denominadas por su localización respecto a la ciudad de Madrid, cada región se caracteriza por tener unos precios de la gasolina ligeramente distintos que además afecta al coste de transporte. Estas cuatro regiones se denominan Norte, Sur, Este y Oeste.

Una vez salen los camiones de estos Fulfillment Centers, se dirigen a los denominados Centros de Distribución o Distribution Centers (DC), que son 7 centros situados en las afueras más próximas de Madrid y que se encargan de dividir los productos por zonas para así ahorrar el máximo de dinero en la última milla. Finalmente, entre estos DCs habrá dos tipos de centros, el primero serán los 2 centros robotizados, que se caracterizan por tener unos costes y un tiempo de procesamiento más bajos, pero, sin embargo, tienen una capacidad limitada, y por lo tanto no puede ir todo el volumen por estos.

Finalmente tenemos los 12 puntos de destino final, que son pequeños comercios o supermercados que solicitan los productos. Estos centros se denominarán End-Point. Claramente, es una simplificación de un sistema real, pero tiene el número de combinaciones que permite entrenar un modelo.

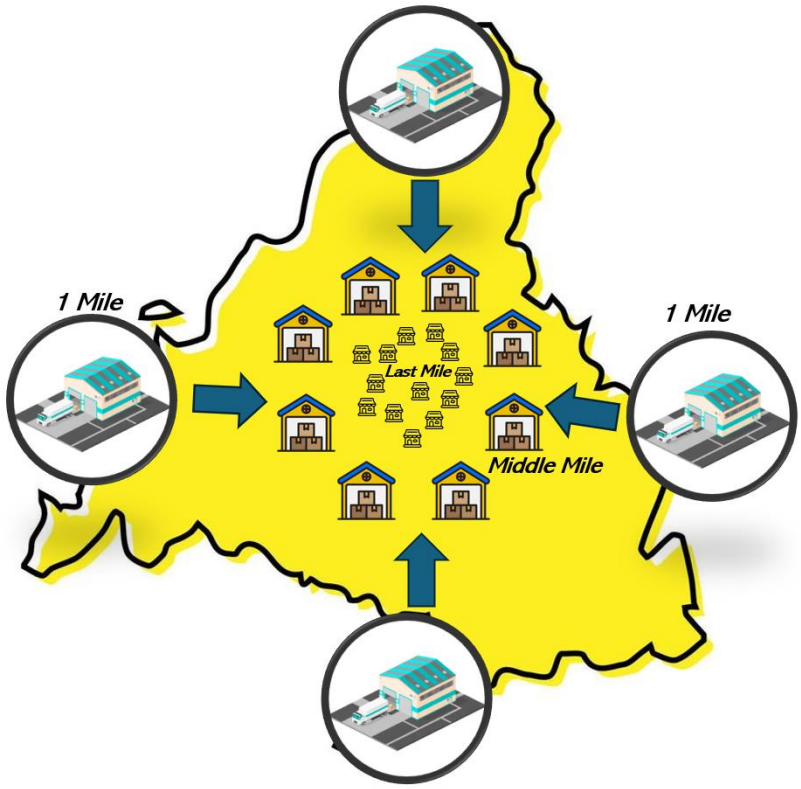



*Ilustración 6. Esquema de la cadena de suministro*

## *4.2 FACTORES PARA TENER EN CUENTA*

Una vez definida la estructura general del problema, habrá que definir los factores que se tendrán en cuenta para obtener la ruta más rentable. En un problema real, quizás existan factores más complejos que no se considerarán en este modelo, ya que la capacidad de obtención de datos es limitada, ya que se generarán sintéticamente.

A continuación, se describirán uno por uno los factores que tendrá en cuenta el algoritmo para minimizar los costes:

- **Distancia entre almacenes:** Claramente, este factor es clave para saber dos cosas, la primera el tiempo que se tarda en transportar el paquete y la segunda, el coste de gasolina que conlleva recorrer esa distancia. Esta distancia es constante, por lo tanto,

se almacenará en una tabla separada ya que no se modificará a no ser que se inscriban nuevos almacenes.

1. **Disponibilidad del producto en los Fulfilment Centers:** Evidentemente, si un determinado tipo de producto no se encuentra en el FC, no se podrá transportar, por lo tanto, hay que tenerlo en cuenta a la hora de hacer los cálculos.
2. **Tiempo de procesamiento de los almacenes:** Este tiempo de procesamiento se supondrá más o menos estable, independientemente de el volumen que estén procesando los almacenes.

- **Coste de procesamiento:** Este tipo de coste es el coste variable que incurren los almacenes cuando procesan un producto. Se tendrá en cuenta un coste medio que se supondrá constante. Hay que destacar que solo se tendrán en cuenta este coste en los centros de distribución debido a que en los End-points no son de la empresa y el coste en los FCs es un coste hundido ya que se ha incurrido en el al almacenar el producto.

3. **Coste del envío:** Para el cálculo de este coste se multiplicará el precio de la gasolina en la región desde la que sale el camión por el número de Km que tiene que recorrer hasta el siguiente punto. Es un coste clave para poder optimizar las rutas al máximo, ya que sino los camiones tenderían a aquellos centros de distribución más baratos de procesamiento independientemente de la distancia.

- **Capacidad de procesamiento:** La capacidad de procesamiento de los centros de distribución es un factor que considerar, ya que quienes son más baratos recibirán más volúmenes provocando una congestión en los almacenes.

# Capítulo 5. ESTRUCTURA DE DATOS

En este capítulo, se definirá la estructura mediante la cual se van a almacenar los datos generados, así como la manera en que se genera y se pueblan las tablas. Estará dividido en dos partes principales, la primera será la realización del DDL y el DML

## *5.1 BASE DE DATOS*

El primer paso es crear la base de datos, para ello, se elaborará un diseño conceptual y posteriormente se creará el esquema lógico. El motivo de la creación de un modelo lógico con varias tablas es que simplifica el problema, ya que hay tablas que se actualizan continuamente, o en las que se añaden nuevos datos, como la tabla encargada de almacenar todos los pedidos, y otras tablas que tienen que modificarse, o actualizarse en un número contado de ocasiones, como la tabla con la información de los almacenes.

### 5.1.1 DISEÑO CONCEPTUAL

Antes de crear una base de datos que contenga una tabla grande, es recomendable dividir el problema en subtablas. De esta manera, el usuario puede acceder a los recursos de manera más sencilla y eficiente. Por ello, se ha desarrollado el siguiente modelo conceptual, que se puede ver en la Ilustración 7.

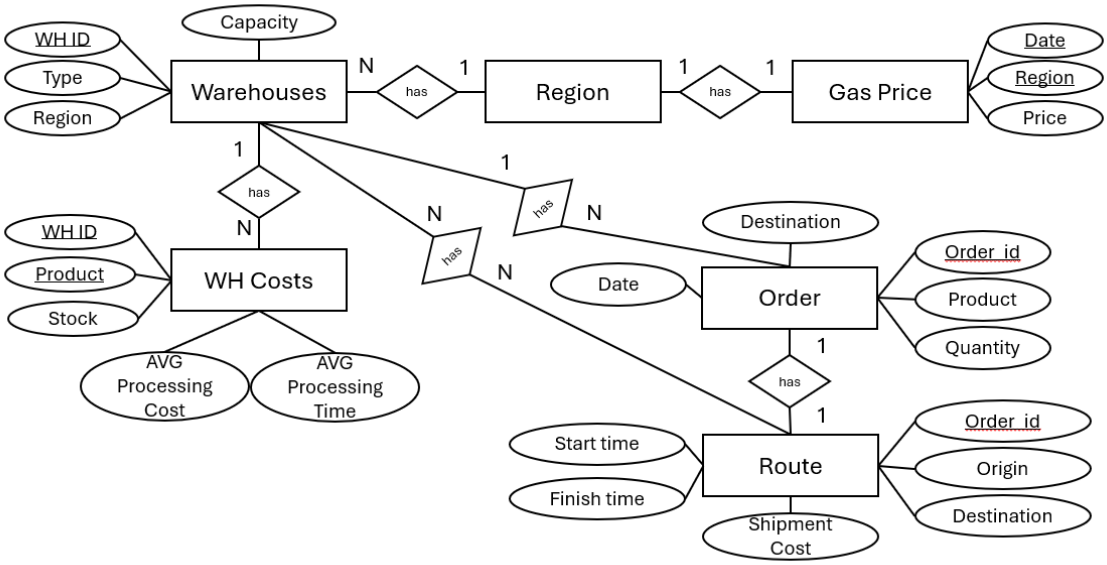



*Ilustración 7. Diseño Conceptual de la base de datos*

En esta ilustración se destaca como un mismo almacén puede tener diversos costes de procesamiento y de tiempo debido al producto. Además, solo se puede tener un producto por pedido debido a que cada producto se ha de transportar en un camión a una temperatura distinta. Por último, se supondrá que cada región tendrá un precio determinado de la gasolina.

### 5.1.2 DISEÑO LÓGICO

Una vez realizado el diseño conceptual, es hora de llevar a cabo un diseño lógico de la base de datos. Este diseño se basará en el desarrollado en el conceptual, pero para simplificar, la tabla de la región será obviada. A continuación, se muestra los resultados en la siguiente Ilustración 8:

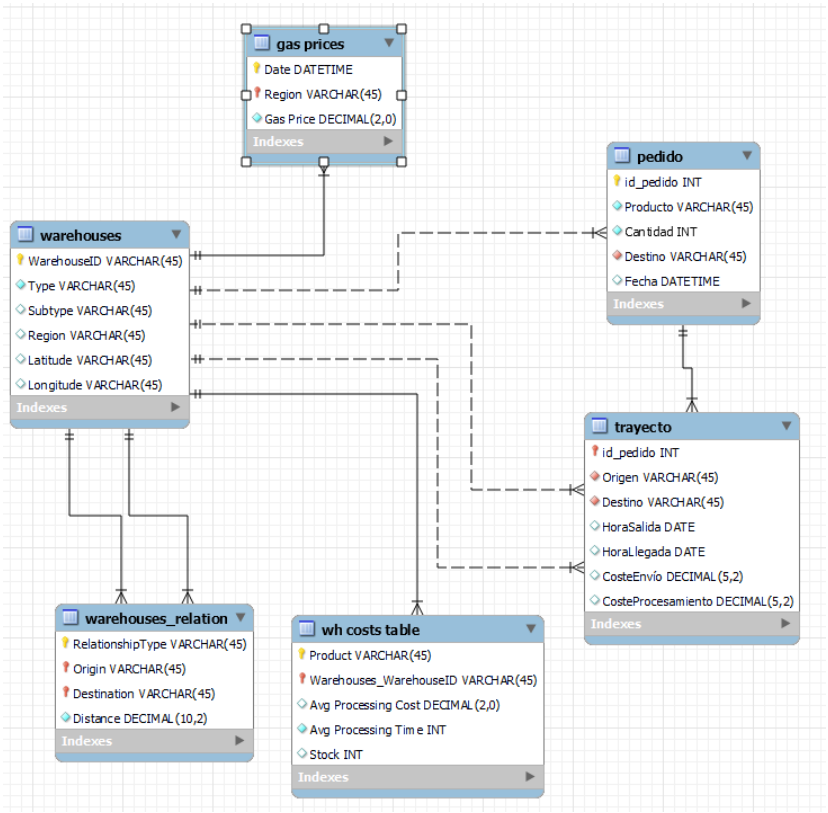



*Ilustración 8. Diseño Lógico de la base de datos*

### 5.1.3 CONEXIÓN DE PYTHON Y MYSQL

Una vez creada la estructura de la base de datos, es hora de conectar la base de datos con Python para poder leer, modificar y poblar las tablas usando código de Python, en vez de manualmente. Para realizar la conexión se necesitan seguir los siguientes pasos:

1. Lo primero de todo es instalar el conector de Python ejecutando el siguiente código en el cmd del sistema:

```
pip install mysql-connector-python
```

2. Una vez instalado el conector, hay que realizar la conexión a la base de datos para ello, hay que crear un Python script y ejecutar el siguiente código:

```
import mysql.connector

conn =  mysql.connector.connect(
    host='localhost',
```

```
    username='root',
    password = 'carlos',
    database='DB_TFM_1'
)

# Crear un cursor
cursor = conn.cursor()
# Ejecutar una consulta
cursor.execute('SELECT * FROM Warehouses')
# Obtener los resultados
results = cursor.fetchall()
for row in results:
    print(row)
# Cerrar el cursor y la conexión
cursor.close()
conn.close()
```

Como se puede apreciar, se tiene que crear un conector con el host de la base de datos, que en este caso ha sido configurada como localhost, con el nombre de usuario, que es el creado por defecto, la contraseña de acceso a mysql y por último el nombre de la base de datos, en este caso llamada 'DB_TFM_1'.

## 5.2 *POBLACIÓN DE LA BASE DE DATOS*

Una vez creadas las tablas y las relaciones entre sí y obtenida la conexión con Python, es hora de poblar estas tablas con los datos que se utilizarán para testear los modelos. En los siguientes puntos se explicará cómo se han completado las tablas.

### 5.2.1 TABLA DE LOS ALMACENES ('WAREHOUSES')

La primera tabla que se puebla es la de los almacenes. Para ello, se van a distinguir entre tres tipos de almacenes, el primero el denominado Fulfillment Center, que consisten en grandes almacenes, habrá cuatro de ellos. El segundo tipo de almacenes son los centros de distribución, también llamados Distribution Centers, de los cuales habrá 7 centros situados en las afueras de Madrid y, por último, los puntos finales de entrega que serán los encargados de realizar los pedidos, que estarán situados en el centro de Madrid y serán 12 establecimientos.

Una vez definidos el número y tipo de los almacenes es la hora de localizarlos en el mapa. Para ello, se colocarán manualmente los cuatro FCs debido a que se busca que estén

localizados estratégicamente en los cuatro puntos cardinales alrededor de Madrid. Para ello, se decide la zona en la que se quiere colocar el centro y se copian las coordenadas en el código de Python. Sin embargo, esta no es la manera óptima de conseguir colocar todos los almacenes, centros de distribución y puntos finales en el mapa, es por eso por lo que a la hora de colocar los otros centros se utilizará otro método.

Este método consiste en elegir las coordenadas del centro de Madrid que son: **Latitud: 40.415347, y Longitud: -3.707371**, y a partir de allí seleccionamos aleatoriamente la localización de los almacenes mediante una distribución uniforme.

Para la distribución uniforme, se necesitan establecer el rango de valores. Para ello, como queremos que los puntos de entrega estén situados en el centro se seleccionará un rango pequeño. Por otro lado, para que haya más probabilidades de que los centros de distribución estén alejados se ampliará el rango de la distribución normal.

Una vez definidas las distribuciones normales, se extraen los resultados y se plotean en la siguiente Ilustración 9:

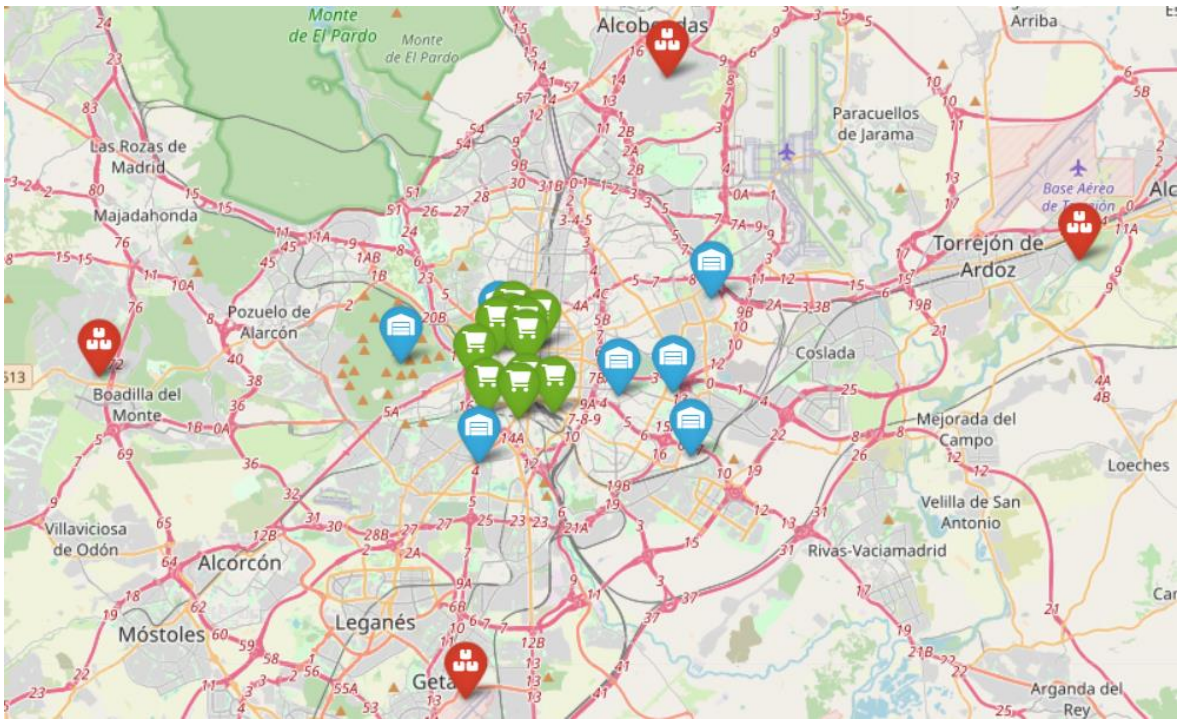



*Ilustración 9. Mapa que contiene la distribución de los almacenes*

Hay que destacar que el mapa ha sido generado diversas veces, hasta alcanzar una distribución de los almacenes adecuada. Esto se consigue cambiando el valor del seed, en este caso el escogido es el seed 0.

```
np.random.seed(0)
```

### 5.2.2 TABLA DE DISTANCIAS ENTRE ALMACENES ('WAREHOUSES_RELATION')

Una vez definidas las coordenadas de todos los puntos, es hora de calcular la distancia entre puntos. Para simplificar el problema, el código calcula la distancia geográfica entre ambos puntos y luego le añade una penalización extra del 30% para simular una distancia real por carretera. Este vector de distancias se obtiene utilizando la fórmula de Haversine, ya que no podemos calcular directamente la distancia euclídea al ser la tierra esférica. Realmente, se podría aproximar mediante la distancia euclídea porque el radio terrestre es de 6371 km mientras que la distancia entre dos puntos del sistema no será mayor a 150 km, aun así, se ha decido usar la fórmula de Haversine que es la siguiente:

$$\Delta Latitud = latitud_2 - latitud_1$$

$$\Delta Longitud = longitud_2 - longitud_1$$

$$a = \sin^2\left(\frac{\Delta Latitud}{2}\right) + \cos(latitud_1) \cdot \cos(latitud_2) \cdot \sin^2\left(\frac{\Delta Longitud}{2}\right)$$

$$c = 2 \cdot \arctan 2\left(\sqrt{(a)}, \sqrt{(1-a)}\right)$$

$$dist_{Km} = 6371Km \cdot c$$

*Ecuación 1. Fórmula de Haversine [11]*

En la siguiente Ilustración 10. Distancias entre distintos almacenes se encuentran las primeras filas de la tabla después de realizar los cálculos ordenadas de mayor a menor distancia:

| RelationshipType | Origin | Destination | Distance |
|---|---|---|---|
| FULL_ESTE TO W3 | FULL_ESTE | W3 | 35.26 |
| FULL_ESTE TO W5 | FULL_ESTE | W5 | 32.59 |
| FULL_OESTE TO W2 | FULL_OESTE | W2 | 31.66 |
| FULL_OESTE TO W7 | FULL_OESTE | W7 | 30.64 |
| FULL_ESTE TO W6 | FULL_ESTE | W6 | 30.06 |
| FULL_OESTE TO W1 | FULL_OESTE | W1 | 29.45 |
| FULL_OESTE TO W4 | FULL_OESTE | W4 | 26.70 |
| FULL_ESTE TO W4 | FULL_ESTE | W4 | 24.63 |
| FULL_SUR TO W2 | FULL_SUR | W2 | 24.09 |
| FULL_ESTE TO W7 | FULL_ESTE | W7 | 22.33 |
| FULL_NORTE TO W5 | FULL_NORTE | W5 | 22.03 |
| FULL_ESTE TO W1 | FULL_ESTE | W1 | 21.97 |
| FULL_OESTE TO W6 | FULL_OESTE | W6 | 20.62 |
| FULL_NORTE TO W3 | FULL_NORTE | W3 | 20.02 |
| FULL_OESTE TO W5 | FULL_OESTE | W5 | 19.96 |
| FULL_NORTE TO W7 | FULL_NORTE | W7 | 19.41 |

*Ilustración 10. Distancias entre distintos almacenes*

Estos son los resultados obtenidos aplicando la fórmula de Haversine para sacar la distancia entre dos puntos cardinales y luego multiplicarla por un 30% para simular que la carretera no constituye una línea recta.

### 5.2.3 TABLA DE LOS PRECIOS DE LA GASOLINA

Esta tabla se irá refrescando cada día, añadiendo así los precios de la gasolina diarios. Para simular los precios de la gasolina se utilizará una distribución normal en la que variarán la media y la desviación típica según la región. De tal forma, que las regiones fluctuarán cambiando diariamente las rutas más baratas y caras. Se definen las siguientes medias y desviaciones típicas:

- **Región Norte:** Precio medio 1.45$, Desviación típica 0.5$
- **Región Sur:** Precio medio 1.5$, Desviación típica 0.3$
- **Región Este:** Precio medio 1.6$, Desviación típica 0.2$
- **Región Oeste:** Precio medio 1.8$, Desviación típica 0.1$

Evidentemente, al estar centrados solo en Madrid los precios de la gasolina variarían menos en un caso real, pero en este caso se simula que con mayor diferencia para ver claramente los precios en los objetos.

### 5.2.4 TABLA DE LOS COSTES DE PROCESAMIENTO Y ALMACENAMIENTO

Para completar la tabla de los costes de almacenamiento se tendrá en cuenta el tipo de almacén, al ser el único coste variable el coste de procesamiento de los centros de distribución son los únicos que se minimizarán. Los costes de procesamiento de los almacenes varían según el tipo de almacén, es por eso, que en la tabla de almacenes se definen dentro de los centros de distribución el subtipo de almacén al cual pertenecen. Hay dos tipos de almacén, el primero son los robotizados, con costes variables más reducidos, y un menor lead time por la alta automatización del proceso, sin embargo, la capacidad de procesamiento de estos almacenes es limitada. El segundo tipo consiste en los almacenes manuales, los cuales se caracterizan por tener un mayor coste de procesamiento por unidad y un mayor lead time. La principal ventaja de estos almacenes es que tienen un menor coste de construcción, y tienen una mayor flexibilidad en su máxima capacidad de procesamiento.

Para la definición de este problema, se supondrán que los costes de procesamiento de una unidad serán menores cuantas más unidades para los almacenes robotizados, mientras que

los costes de procesamiento de los almacenes manuales disminuirán en menor medida. Además, se considerará que el lead time es aproximadamente un 30% menor para los almacenes robotizados y que la variabilidad es menor. Mientras que para los almacenes manuales tendrán una mayor variabilidad en el tiempo medio de procesamiento.

- **Coste de procesamiento:**
- Almacén robotizado: Para los almacenes robotizados se creará una función que calcule el coste de procesamiento en función del volumen que procese el almacén.
- Almacén manual: Estos costes serán más constantes debido a la flexibilidad que existe a la hora de aumentar o disminuir la capacidad de este tipo de almacenes, por lo tanto, los costes de procesamiento se supondrán más estables.
- **Tiempo de procesamiento:**
- Almacén robotizado: Los tiempos de procesamiento serán constantes independientemente de la capacidad procesada porque al estar los procesos automatizados, no se genera caos cuando ocurren aumentos de volúmenes.
- Almacén manual: En este caso, los volúmenes sí que influirán en el tiempo de procesamiento, ya que el aumento de volúmenes generará un mayor caos en el sistema que provocará retrasos.

### 5.2.5 TABLA DE LOS PEDIDOS

La tabla de los pedidos constará de aquellos pedidos que se han realizado durante los últimos 2 años, esta tabla constará de varias columnas, en las cuales se almacenarán los datos del end-point el cual ha mandado la orden de compra, la fecha en la que la dio, el tipo y la cantidad de producto solicitado.

### 5.2.6 TABLA DE LAS RUTAS

En esta tabla es donde se escribirán las rutas generadas por la inteligencia artificial. Al rellenar la tabla, se crearán las rutas aleatorias, para que el modelo obtenga datos pasados y elabore las mejores rutas posibles. En esta tabla, se almacenarán los costes que se han incumbido en el proceso, así como la hora de salida y de la entrega del producto.

## 5.3 DEFINICIÓN DEL MODELO DE IA

En este capítulo, se busca estudiar cuál de los distintos algoritmos de inteligencia artificial para encontrar la ruta óptima es mejor. Para ello, se hará una exploración profunda de los distintos algoritmos con sus ventajas e inconvenientes, y finalmente, se decidirá aquel que se va a aplicar. Una vez seleccionado el modelo, se procederá al desarrollo de este mediante código de Python y posteriormente, se analizarán los resultados. El objetivo es que el modelo funcione adecuadamente y sea capaz de encontrar la ruta óptima para el caso en el que la interfaz no esté en funcionamiento.

### 5.3.1 TIPOS DE ALGORITMOS

En esta sección, se estudiarán las distintas opciones que existen a la hora de resolver el problema. Se identificarán las ventajas y desventajas de cada modelo a nivel teórico y posteriormente se procederá aquel algoritmo que presente unas características adecuadas para enfrentarse a este modelo.

#### *5.3.1.1 Complejidad del problema*

Antes de estudiar que tipos de soluciones se pueden aplicar al modelo, hay que entender cuáles son los distintos tipos de problemas y su complejidad a la hora de afrontarlos en el ámbito de la ciencia de la computación teórica. [13]

Para ello, lo primero de todo es definir qué significa el tiempo polinómico. El tiempo polinómico es un concepto fundamental en la teoría de la complejidad computacional que se refiere a la cantidad de tiempo que un algoritmo toma para resolver un problema, en función del tamaño de la entrada del problema, donde este tiempo está acotado por una función polinómica.

Un algoritmo se dice que tiene tiempo de ejecución polinómico **(P)** si su tiempo de ejecución T(n) puede expresarse como una función polinómica del tamaño de la entrada $n$. Formalmente, esto significa que existe un polinomio $p(n)$ tal que:

$$T(n) = O\big(p(n)\big) = O(n^k)$$

Donde sí k = 0 es un tiempo constante independientemente del tamaño de la entrada, sería el caso de buscar un elemento en una lista. Si k = 1 es de tiempo lineal, es decir el tiempo de ejecución crece linealmente con el tamaño de la entrada. Si k >=2 el tiempo de ejecución crecerá exponencialmente según el tamaño de la entrada. K es constante independientemente del tamaño de entrada.

Luego existen los tiempos no polinómicos **(NP)**, que son aquellos algoritmos que no pueden ser resueltos en tiempo polinómico, pero, sin embargo, sí que se pueden verificar. Estos algoritmos siguen la siguiente relación complejidad-tiempo:

$$T(n) = O(k^n)$$

Donde k es una constante mayor que 1 y n es el tamaño de la entrada. Tal como se ha comentado en el anterior párrafo, aunque la resolución del problema sea en tiempo no polinómico, la solución se puede verificar de manera rápida en tiempo polinómico.

Evidentemente, con el tiempo NP, a medida que aumente le entrada aumentará considerablemente el tiempo de ejecución. [14]

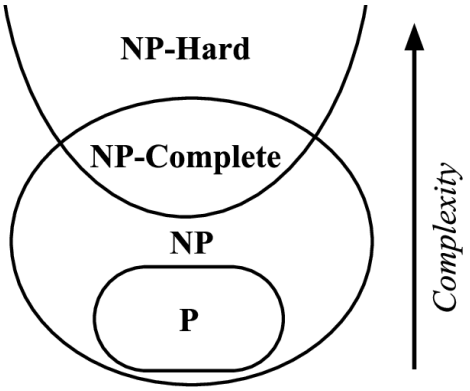



*Ilustración 11. Diagrama de intersección entre las clases P, NP, NP-completa y NP-Difícil [15]*

Como se puede ver en la ilustración, **NP-completos** son los problemas más difíciles dentro de NP, y encontrar una solución rápida para uno de ellos resolvería rápidamente todos los problemas en NP.

**NP-difíciles** incluyen a los problemas NP-completos y otros problemas que pueden no estar en NP y que son, por lo tanto, más difíciles de abordar tanto en términos de resolución como de verificación.

El TSP puede clasificarse de manera diferente dependiendo de cómo se plantee el problema:

- **TSP como Problema de Decisión:** Es NP-completo porque podemos verificar en tiempo polinómico si una ruta dada cumple con un coste específico y cualquier problema en NP se puede reducir a esta versión de TSP en tiempo polinómico.
- **TSP como Problema de Optimización:** Es NP-difícil porque encontrar la ruta de menor coste es un problema complejo que puede no ser verificable en tiempo polinómico y es al menos tan difícil como cualquier problema en NP.

Esta dualidad en la clasificación refleja la complejidad y el desafío inherente del TSP en la teoría de la complejidad computacional.

#### *5.3.1.2 El problema del agente viajero*

El problema de encontrar la mejor ruta para minimizar costes no es un problema nuevo. En el año 1954 fue cuando se publicó la primera solución a este problema, los autores de esta hazaña fueron George Dantzig, Ray Fulkerson, y Selmer Johnson que publicaron un método que solucionaba el Problema del agente viajero (PAV) o más conocido como su nombre en inglés como TSP – Travel Sailsman Problem. En esta publicación titulada "Solutions of a large scale traveling salesman problema" se buscaba encontrar el camino más corto entre 49 ciudades donde un viajero las visitaba una vez y luego retornase al punto de salida. [12]

Aunque su enunciado es sencillo, resolverlo de manera eficiente para un gran número de ciudades es extremadamente complejo debido a la naturaleza combinatoria del problema. Como se ha comentado antes, la primera solución efectiva fue desarrollada por los tres autores de la publicación, y se basaba en el uso de técnicas de programación lineal para

abordar este problema difícil. Esta solución, a pesar de ser inicial, sentó las bases para muchos métodos modernos y avanzados que se utilizan en la actualidad.

Las variantes de la solución a este problema se usan en todo tipo de situaciones en las que se requiera la selección de distintos nodos para llegar a un punto de destino, en los cuales se podría encontrar los problemas de reparto de productos, del número de movimientos que tiene que hacer el robot para llegar a una posición en concreto, entre otros.

#### *5.3.1.3 Soluciones al problema del agente viajero*

A continuación, se describen varias categorías de soluciones y métodos que se han desarrollado para abordar el TSP:

1. Fuerza bruta: Este sistema consiste en calcular todas las combinaciones posibles y luego elegir aquella que suponga un menor coste. Es una solución poco óptima e inaplicable cuando se aumenta considerablemente el número de nodos.
2. Soluciones exactas: Este tipo de soluciones se encargan de buscar la solución óptima garantizada. Buscan una mayor rapidez a la hora de buscar el óptimo.
3. Soluciones heurísticas: Son métodos que obtienen soluciones a una mayor rapidez que las anteriores, sin embargo, corren el riesgo de encontrar un mínimo local. [12]

Entre los posibles algoritmos que se pueden usar para resolver este problema se consideran las siguientes:

- Algoritmos exactos:
    - ***Fuerza bruta:*** valúa todas las posibles permutaciones de las ciudades y selecciona la que tiene el menor costo total.
        - Complejidad: O(n!)
        - Desventaja: Inaplicable cuando hay un número grande de almacenes.
    - ***Dynamic Programming:*** Resuelve el TSP utilizando una tabla que almacena subproblemas más pequeños y combina sus soluciones.
        - Reduce la complejidad a $\boldsymbol{O(n^2 \cdot 2^n)}$

        - Desventajas: Aunque sea exponencial, es más eficiente que el anterior.
    - ***Integer Linear Programming – ILP:*** Representa el TSP como un problema de programación lineal entera y utiliza técnicas de relajación y cortes para encontrar la solución óptima.
        - Desventajas: Puede llegar a tener una alta complejidad dependiendo del tamaño y la estructura del problema.
- Algoritmos heurísticos
    - ***Nearest-Neighbour Algorithm:*** Algoritmo de decisión donde elige una ciudad y va visitando la más cercana.
        - Complejidad: $\boldsymbol{O(n^2)}$
        - Desventaja: Puede encontrar el camino subóptimo.
    - ***Heurística de 2-Opt y k-Opt:*** Esta técnica busca mejora el camino comparando con los k nodos más cercanos a el punto para encontrar aquel óptimo. Mejora iterativamente una ruta inicial hasta que no se puedan realizar más mejoras.
        - Complejidad: $\boldsymbol{O(n^2)}$ por intercambio en el caso de 2-Opt.
    - ***Algoritmos Genéticos:*** Utilizan principios de la evolución y genética, como la selección, cruzamiento y mutación, para generar nuevas soluciones a partir de una población de soluciones existentes.
        - Complejidad: Depende de la configuración, típicamente **O(population size × generations × n).**
    - ***Simulated Annealing:*** Permite aceptar soluciones peores temporalmente para escapar de óptimos locales y encontrar una mejor solución global.
        - Complejidad: Depende del número de iteraciones, típicamente **O(iterations × n).**
    - ***Ant Colony Optimization:*** Simula el comportamiento de las hormigas que depositan feromonas para indicar rutas prometedoras, donde las soluciones se construyen probabilísticamente favoreciendo las rutas con mayor depósito de feromonas.

- Complejidad: Depende de la configuración, pero generalmente $\boldsymbol{O(iterations \times n^2)}$.

- ***Cross-Entropy algorithm:*** Es un algoritmo ampliamente usado para resolver problemas del tipo NP-díficil, como es el caso del TSP. Es un algoritmo iterativo que busca minimizar una función. Se caracteriza por su capacidad para manejar múltiples criterios y restricciones que lo hace ideal para escenarios de minimización más complejos como es el caso de este proyecto. [16]
  - Complejidad: La complejidad de este algoritmo depende también de su configuración, pero a modo general será desarrollada en el siguiente apartado.

#### *5.3.1.4 Algoritmo seleccionado (Cross-Entropy Algorithm)*

En esta sección se justificará el algoritmo seleccionado, se explicarán las causas de la selección y, por último, se procederá a explicarlo más profundamente.

En primer lugar, tras realizar una búsqueda exhaustiva de los posibles algoritmos que se pueden emplear a la hora de resolver un problema de creación de rutas de una cadena de suministro, se llegó a la conclusión de que hay dos planteamientos. En primer lugar, están los algoritmos exactos, donde la ruta que se obtiene es la ruta óptima, aunque su principal desventaja es que al aumentar la complejidad del problema necesita mayor potencia computacional y mayor tiempo para llegar a la solución, lo cual puede ser inviable. En segundo lugar, se encuentran los algoritmos heurísticos, que son aquellos que no son capaces de encontrar la solución óptima con total seguridad, pero, sin embargo, consiguen una solución subóptima en un tiempo menor.

Para la resolución de este problema se ha seleccionado un algoritmo metaheurístico, cuya característica principal es que se aplican reglas más flexibles que un algoritmo heurístico lo que le da más flexibilidad a la hora de evitar óptimos locales y, por tanto, tienen una mayor generalidad. En este caso, el algoritmo seleccionado es el Cross-Entropy Algorihtm. [17]

##### **5.3.1.4.1 Motivos de selección del modelo**

Los motivos por los cuales se ha elegido este tipo de algoritmo frente a los anteriormente expuestos son los siguientes. Aunque antes de numerarlos, hay que tener en cuenta que el objetivo de este trabajo no es hacer una comparativa de distintos algoritmos para determinar cual es el más adecuado para nuestro problema, sino que, mediante la búsqueda de un estudio ya realizado para un problema similar, se pueda determinar cuál de ellos es el adecuado.

Es por eso por lo que gracias a el estudio del comportamiento de los siguientes tres algorimtos Genetic Algorithm, Simulated Annealing y Cross-Entropy llevado a cabo por Behnam Fahimniaa, Hoda Davarzani y Ali Eshragh en el artículo 'Planning of complex supply chains: A performance comparison of three meta-heuristic algorithms' publicado en octubre de 2015. Donde se estudia para un problema real de una cadena de suministro de un fabricante australiano de ropa, para encontrar la ruta que minimizase al máximo el coste de transporte, así como las emisiones de C02 emitidas a la atmosfera. [16]

La similitud del problema es elevada, ya que como se puede ver en la Ilustración 12, las similitudes son notables.

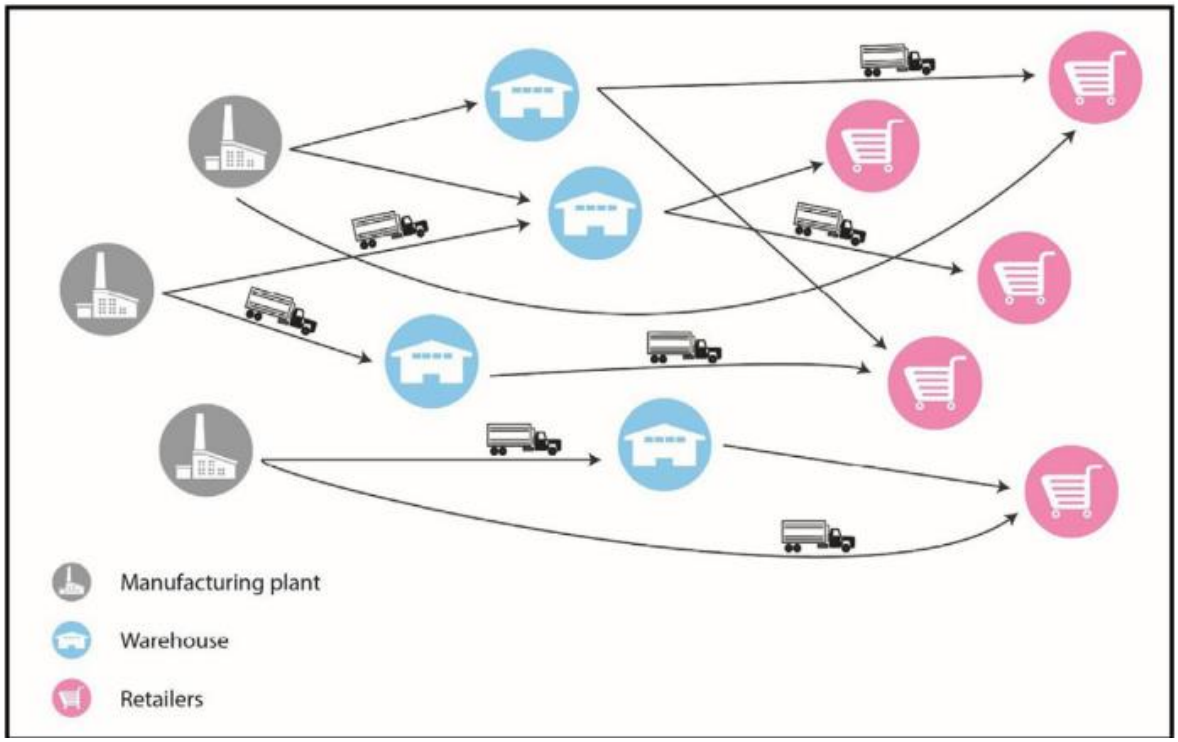



*Ilustración 12. Topología de la cadena de suministro del fabricante de ropa australiano. [16]*

Los resultados de este artículo son muy claros, para una red de distribución con tres plantas, cuatro almacenes y cinco clientes, además de cinco productos y seis máquinas por fábrica, el algoritmo que mejor funcionaba era el de Cross-Entropy (CE) que conseguía obtener en el menor tiempo de ejecución los menores costes del sistema, así como las menores emisiones, tal y como se puede ver en la Ilustración 13:

Characteristics of the best found solutions using the three meta-heuristic algorithms

| Algorithm | Time (s) | Cost components ($) | | | | | |
|---|---|---|---|---|---|---|---|
| | | Total cost | Procurement/Outsourcing | Manufacturing | Distribution | Emissions | Shortage |
| GA | 28,793 | 10,697,703 | 5,966,952 | 2,422,112 | 1,656,440 | 348,250 | 303,949 |
| SA | 22,640 | 10,711,019 | 5,976,916 | 2,445,300 | 1,662,629 | 355,458 | 270,716 |
| CE | 18,605 | 10,671,815 | 5,960,082 | 2,502,832 | 1,606,092 | 323,375 | 279,434 |

*Ilustración 13. Tabla comparativa de resultados según el modelo utilizado[16]*

Es por eso por lo que, debido a la similitud del problema, se escoge el modelo que mejor a funcionado para este caso.

### 5.3.1.4.2 Desarrollo teórico del modelo

El algoritmo Cross-Entropy (CE) es un algoritmo metaheurístico que se caracteriza por su capacidad de generalización, lo que le permite ser usado en una gran cantidad de problemas distintos de tipo NP-difícil. Este algoritmo fue desarrollado en 1997 por Rubistein para estimar la probabilidad de eventos improbables, aunque posteriormente este algoritmo se empezó a usar en otros campos como en el diseño de cadenas de suministro o rutas de vehículos. [18] [19]

El funcionamiento del algoritmo es el bastante parecido al del 'Ant Colony Optimization', ya que pertenecen a los modelos basados en la búsqueda denominados en inglés como 'model-based search'. Este tipo de algoritmos aplica principios de modelización probabilísticos, refinando iterativamente su búsqueda hasta llegar a las soluciones óptimas. He aquí una explicación general de su funcionamiento:

Lo primero que realiza el algoritmo es definir una distribución probabilística que servirá para generar un conjunto de soluciones probables. Este conjunto inicialmente será muy amplio debido a la elevada incertidumbre que aparece cuando se inicia la búsqueda.

Una vez se ha actualizado la distribución probabilística, el algoritmo creará un determinado número de puntos sacados de la función probabilística. Cada una de estas representaciones, se encargará de explorar una parte del espacio que tiene alrededor, con el fin de buscar a la mejor solución. Después de explorar el espacio, se evalúan los resultados mediante la introducción de estos puntos en la función que hay que minimizar, para así determinar como de buena o mala es la solución aplicada.

A continuación, se escogen las mejores soluciones entre los puntos anteriores. Estas soluciones solo representarán una función pequeña de las muestras totales que se habían ejecutado previamente. Por último, se actualizarían los parámetros que definen a la distribución que se ha definido en el primer paso, provocando que esta sea menos extensa, y por tanto acote el rango de búsqueda. Este proceso se repetirá de forma iterativa hasta que se converja en la solución final.

#### 5.3.1.4.3 Desarrollo del modelo matemático

En el modelo completo incluyendo el que integra el modelo LLM para que los usuarios puedan interactuar directamente con el sistema sigue la siguiente función objetiva:

*Ecuación 2. Función objetivo a minimizar*

$$C = C_1 + \alpha * \beta_i * C_2$$

Donde C representa el coste total del sistema, y la función a minimizar. Estará integrada por tres componentes, la primera $C_1$ representa el coste económico del sistema, es decir, el coste de procesamiento, el coste de transporte. En el segundo componente $C_2$ representa la función del tiempo que tarda en llegar el pedido, es decir mide la velocidad de entrega al cliente. Esta velocidad se medirá con un coste asociado al tiempo que tarda representado por $\alpha$ [$/h] que se determinará en función del producto. Finalmente, se representa el coste del modelo LLM con $\beta_i$, el valor de beta vendrá determinado por la última interacción que ha tenido un usuario de un almacén i. Es decir, si la última interacción del almacén o delivery station es buena será 0, si es mala su peso aumentará y en el caso de que sea muy mala, por ejemplo, una rotura de una máquina, el valor será infinito para indicar que no se puede procesar ningún volumen.

A continuación, se desarrollan los dos parámetros en detalle:

Lo primero es definir los índices que se van a usar:

*Índice del producto:* $i \in 1,2,3,4$

*Índice del Fulfilment center:* $f \in 1,2,3,4$

*Índice del Centro de Distribución:* $d \in 1,2,3,\dots 7$

*Índice del Punto Final:* $e \in 1,2,3,\dots 12$

*Índice del tiempo:* $t \in 1,2,3,\dots T$

*Índice de la región:* $r \in 1,2,3,4$

*Índice del cliente:* $c \in 1,2,3\dots, C$

Una vez definidos los índices hay que definir los parámetros del sistema:

$co$ *Consumo de los camiones en (l/km) – es una constante de 5 l/km*

$d_{fd}$ *Distancia entre los almacenes f a d*

$gp_{rt}$ *Precio de la gasolina en la región r en el tiempo t*

$n_{idt}$ *Número de unidades procesadas del producto i, en la delivery station d en el momento t*

$pc_{dt}$ *Coste de procesamiento (Processing Cost) de la delivery station d en el tiempo t*

$v$ *Velocidad media del camión (km/h) – es una constante de 10 km/h*

$u_{idt}$ *Tiempo de procesamiento del producto i en la delivery station d en el momento t*

$\beta_{idt}$ *Valor determinado por el LLM del producto i, deliver station D en el momento t*

$\alpha_{id}^{max}$ *Máximas unidades del producto i que se pueden procesar en la deliver station D*

Las variables continuas son las siguientes:

$X_{ift}$ *Inventario del producto i en el gran almacén f en el momento t*

$Y_{idt}$ *Inventario del producto i en el centro de distribución d en el momento t*

$Q_{ift}$ *Reposición del producto i en el gran almacén f en el momento t – unidades que entran*

Lo siguiente es desarrollar la Ecuación 2, para ello se divide la explicación en sus dos componentes principales, $C_1$ $y$ $C_2$. A continuación, se explica la primera:

$C_1$ es la componente que representa los gastos económicos que se incurren en el sistema, es decir el gasto de transporte y el gasto de procesamiento. Las primeras dos componentes de la Ecuación 3 son los costes incurridos a la hora de transportar las mercancías de un almacén a otro. Para calcular este gasto se necesita saber el precio de la gasolina en esa región determinada y el consumo de los camiones, así como la distancia entre los almacenes. Finalmente, la tercera componente de la ecuación representa el coste de procesamiento del producto en la delivery station, este coste se calcula como el número de unidades a procesar en el pedido por el coste de procesamiento en ese determinado almacén.

*Ecuación 3. Desarrollo de la componente uno*

$$C_1 = \sum_f \sum_d \sum_r \sum_t gp_{rt} * co * d_{fd} + \sum_d \sum_e \sum_r \sum_t gp_{rt} * co * d_{de} + \sum_e \sum_c \sum_r \sum_t gp_{rt} * co * d_{de} + \sum_d \sum_i \sum_t n_{idt} * pc_{dt}$$

En el caso del tiempo a minimizar tiene tres componentes, las dos primeras son los tiempos de transporte entre almacenes, y la última es el tiempo de procesamiento. Como se puede ver, la beta se ha incluido solo en el tiempo de procesamiento de los almacenes para un determinado producto en un determinado almacén y en un tiempo. Luego se multiplica por el tiempo de procesamiento de una unidad y finalmente por el número de unidades del pedido.

*Ecuación 4. Componente dos de la función objetivo*

$$C_2 = \sum_f \sum_d \frac{d_{fd}}{v} + \sum_d \sum_e \frac{d_{de}}{v} + \sum_e \sum_c \frac{d_{ec}}{v} + \sum_i \sum_d \sum_t \beta_{idt} * u_{idt} * n_{it}$$

Una vez ya definida y desarrollada la ecuación a minimizar, es hora de definir las restricciones del sistema.

*Ecuación 5*

$$X_{ift} = X_{if(t-1)} + Q_{ift} - \sum_d n_{ifdt}$$

*Ecuación 6*

$$Y_{idt} = Y_{id(t-1)} + \sum_f n_{ifdt} - \sum_e n_{idet}$$

*Ecuación 7*

$$Y_{idt} \leq \alpha_{id}^{max}$$

*Ecuación 8*

$$0 \geq Y_{idt}, 0 \geq X_{ift}$$

*Ecuación 9*

$$\delta_{ifdt}^{min} \leq n_{ifdt} \leq \delta_{ifdt}^{max}$$

*Ecuación 10*

$$\gamma_{idet}^{min} \leq n_{ifdt} \leq \gamma_{idet}^{max}$$

La Ecuación 5 es la restricción que representa el volumen que se encuentra en cada momento en el gran almacén, representa lo mismo, pero para el centro de distribución la Ecuación 6. La Ecuación 7 representa que el volumen que tiene el centro de distribución d del producto i en el tiempo t, tiene que ser menor o igual a la capacidad máxima de ese almacén d para ese producto i. Finalmente, la Ecuación 8 define que no pueden haber menos de cero unidades en el almacén, mientras que la Ecuación 9 y la Ecuación 10 sirven para indicar el mínimo y el máximo de cada producto que pueden llevar los camiones.

# Capítulo 6. DESARROLLO DEL MODELO DE CE

Una vez planteadas las ecuaciones matemáticas es momento de desarrollar el modelo en Python. Para ello, se realizarán dos modelos. El primero será un modelo más simple donde no se implementará la parte del LLM, sino que solo se utilizará el modelo de Cross-Entropy. A continuación, se incrementará la dificultad del modelo añadiéndole un modelo LLM, el cuál será capaz de interpretar y procesar los mensajes que mandan los jefes de los distintos almacenes y centros de procesamiento que conforman la red de suministro. Una vez interpretado el mensaje se realizarán los cambios de los parámetros correspondientes para el modelo principal.

## *6.1 IMPLEMENTACIÓN DEL MODELO EN PYTHON*

El código proporcionado en el Anexo A está diseñado para optimizar la cadena de suministro mediante el uso de un algoritmo de entropía cruzada o Cross-Entropy. Para lograr este objetivo, el código se compone de dos funciones principales: `calculate_total_cost` y `cross_entropy_optimization`. La primera función se encarga de calcular el costo total asociado a una combinación específica de centro de cumplimiento (FC), estación de distribución (DS) y punto final (END) para un cliente y un producto determinado. La segunda función implementa el algoritmo de Cross-Entropy para minimizar dicho costo total seleccionando la mejor combinación posible de estos tres elementos.

El código inicia con la importación de las bibliotecas necesarias: `numpy` para operaciones numéricas, `pandas` para manejo de datos y un módulo específico `DataExtractionModule` que se encarga de la extracción de datos relevantes del sistema. Este último módulo, contiene todas las funciones necesarias para poder extraer la información de la base de datos del modelo, y puede ser encontrado en el Anexo B.

### 6.1.1 DESCRIPCIÓN DE LA FUNCIÓN DE CÁLCULO DE COSTES

#### *6.1.1.1 Parte I: Extracción de información de la base de datos*

La primera función `calculate_total_cost` recibe varios parámetros que incluyen el centro de cumplimiento, la estación de distribución, el punto final, las coordenadas del cliente, el producto a entregar y la cantidad del producto. Esta función comienza extrayendo el stock y la capacidad máxima tanto del centro de cumplimiento como de la estación de distribución mediante llamadas a funciones del módulo `dataExtractionModule`. Luego, verifica si se cumplen las condiciones definidas en el modelo matemático definido anteriormente. Estas condiciones son las siguientes: el producto está disponible en el centro de cumplimiento y si la capacidad de la estación de distribución es suficiente para manejar la cantidad solicitada. Si alguna de estas condiciones no se cumple, la función retorna un valor infinito, indicando que la combinación no es viable.

A continuación, la función calcula las distancias de envío entre los distintos puntos: desde el centro de cumplimiento a la estación de distribución, desde la estación de distribución al punto final y desde el punto final hasta el cliente. Estas distancias son extraídas de la base de datos utilizando funciones del módulo `dataExtractionModule`. Seguidamente, se extraen los costos de procesamiento y los precios de la gasolina para cada segmento del recorrido. El costo de procesamiento y los tiempos de procesamiento asociados con la estación de distribución y el producto también se obtienen del mismo módulo.

#### *6.1.1.2 Parte II: Cálculo del coste total estimado*

##### 6.1.1.2.1 Modelo sin aplicación del LLM

Con toda esta información, la función procede a calcular el costo de transporte y procesamiento. El costo de transporte se basa en las distancias y los precios de la gasolina, mientras que el costo de procesamiento se multiplica por la cantidad de producto. Adicionalmente, se calcula el tiempo de transporte y procesamiento, considerando una velocidad promedio de viaje y el tiempo de procesamiento por unidad de producto.

Finalmente, el costo total se obtiene sumando el costo de transporte y procesamiento con un factor de costo de tiempo. La función retorna este costo total redondeado a cuatro decimales.

A continuación, se muestran el código de las ecuaciones implementadas:

La componte 1:

```
c_1 = round((shipping_distance_fc_to_ds * gas_price_fc_ds * consumption +

            shipping_distance_ds_to_end   *   gas_price_ds_end   *   consumption   +
gas_price_end_client*consumption*shipping_distance_end_to_client +

       processing_cost * quantity),5)
```

La componte 2:

```
# Avg speed of the trip

speed = 10  # km/h

# Computing the component C2 of the objective function, units in hours

c_2 = round((shipping_distance_fc_to_ds / speed +

       shipping_distance_ds_to_end / speed + shipping_distance_end_to_client/speed
+

       quantity * processing_time),5)
```

Finalmente, se calcula la suma de las dos componentes. Hay que destacar que la componente 2 da su resultado en horas, mientras que la componente 1 lo da en euros, por lo tanto, la componente 2 se tiene que multiplicar por una constante que expresa el coste del tiempo en euros. En este caso en concreto este valor es uno, pero posteriormente se realizará un estudio de sensibilidad.

```
# Alpha is a constant expressing the cost of time in euros

alpha = 1  # eur/hour
```

```
total_cost = c_1 + alpha * c_2

return round(total_cost, 4)
```

#### 6.1.1.2.2 Modelo con la aplicación del LLM

Una vez definido el modelo de CE, es hora de introducir la componente de LLM para que los distintos almacenes puedan actualizar el estado en tiempo real de estos. Para ello se utilizará y entrenará un modelo de inteligencia artificial que sea capaz de interpretar la frase introducida por el mánager del almacén en el sistema.

Una vez el mánager haya introducido la frase indicando el estado del almacén se actualizará en la base de datos la capacidad de ese almacén. La actualización se realizará en la tabla de la modelo llamada 'warehouses' y en la columna 'Availability'. El modelo LLM devolverá un número del 1 al 3, donde 1 indica que el almacén funciona correctamente, 2 indica que existen problemas a la hora de funcionar y 3 sirve para indicar que el almacén está estropeado o que a alcanzado su capacidad máxima.

Una vez actualizados se introducirá en el código del modelo una nueva variable llamada 'beta' que será la encargada de representar el estado del almacén. Esta variable solo multiplicará a la componente 2 ('c_2' del código) debido a que es en esta componente donde se tiene en cuenta el tiempo de procesamiento del almacén.

```
total_cost = c_1 + alpha * c_2 * beta
```

En el siguiente capítulo, Desarrollo del Modelo LLM se desarrollará como se ha entrenado el modelo LLM.

### 6.1.2 FUNCIÓN QUE IMPLEMENTA EL MODELO DE CROSS-ENTROPY

La segunda función, `cross_entropy_optimization`, se encarga de realizar la optimización mediante el algoritmo de Cross-Entropy. Esta función recibe listas de centros de cumplimiento, estaciones de distribución y puntos finales disponibles, así como las coordenadas del cliente, el producto y la cantidad a entregar. Además, permite configurar el

número de muestras a generar en cada iteración, la fracción de muestras consideradas élite y el número máximo de iteraciones para el proceso de optimización.

### *6.1.2.1 Funcionamiento del algoritmo de CE*

El algoritmo comienza inicializando distribuciones de probabilidad uniformes para la selección de centros de cumplimiento, estaciones de distribución y puntos finales. En cada iteración, se generan múltiples muestras aleatorias de estas combinaciones utilizando las distribuciones de probabilidad actuales. Para cada muestra, se calcula el costo total utilizando la función `calculate_total_cost`. Las muestras se ordenan según el costo y se selecciona una fracción de las mejores muestras (muestras élite).

A continuación, se mostrará con imágenes un ejemplo de funcionamiento. En primer lugar, se inicializan las distribuciones:

```
Probabilites distribution in itertation: 0 : [0.25 0.25 0.25 0.25]
Probabilites distribution in itertation: 0 : [0.14285714 0.14285714 0.14285714 0.14285714 0.14285714 0.14285714
 0.14285714]
Probabilites distribution in itertation: 0 : [0.08333333 0.08333333 0.08333333 0.08333333 0.08333333 0.08333333
 0.08333333 0.08333333 0.08333333 0.08333333 0.08333333 0.08333333]
```

*Ilustración 14. Inicialización de la distribución de probabilidades*

El siguiente paso es calcular los costes para las distintas combinaciones:

```
[('FULL_NORTE', 'W6', 'FIN04', 1199.8154), ('FULL_SUR', 'W7', 'FIN07', 1235.06), ('FULL_NORTE', 'W7', 'FIN08', 1236.8956), ('FULL_NORTE', 'W7', 'FIN08', 1236.8956), ('FULL_NORTE', 'W7', 'FIN05', 1245.4301), ('FULL_ESTE', 'W6', 'FIN08', 1301.2731), ('FULL_SUR', 'W2', 'FIN07', 1360.9305), ('FULL_SUR', 'W2', 'FIN05', 1364.9946), ('FULL_NORTE', 'W4', 'FIN10', 1408.7436), ('FULL_SUR', 'W4', 'FIN12', 1416.9522), ('FULL_OESTE', 'W2', 'FIN05', 1457.8336), ('FULL_SUR', 'W5', 'FIN06', 1458.2665), ('FULL_ESTE', 'W4', 'FIN09', 1481.5987), ('FULL_ESTE', 'W4', 'FIN01', 1481.7571), ('FULL_NORTE', 'W5', 'FIN03', 1508.6146), ('FULL_NORTE', 'W5', 'FIN11', 1510.3906), ('FULL_SUR', 'W1', 'FIN10', 1517.0395), ('FULL_SUR', 'W1', 'FIN04', 1522.6244), ('FULL_OESTE', 'W4', 'FIN03', 1534.9701), ('FULL_NORTE', 'W5', 'FIN06', 1535.046)]
Iteration 0: Best cost = 1199.8154
```

*Ilustración 15. Resultados de la primera iteración*

Y una vez obtenidos los resultados, se actualizan las probabilidades.

```
Probabilites distribution in itertation: 1 : [0.1 0.5 0.  0.4]
Probabilites distribution in itertation: 1 : [0.  0.2 0.  0.2 0.  0.2 0.4]
Probabilites distribution in itertation: 1 : [0.  0.  0.  0.1 0.2 0.  0.2 0.3 0.  0.1 0.  0.1]
```

*Ilustración 16. Nueva distribución de probabilidades*

A partir de las muestras élite, se actualizan las distribuciones de probabilidad para cada elemento, basándose en la frecuencia con la que cada centro de cumplimiento, estación de distribución y punto final aparece en las muestras élite. Este proceso de actualización se repite en cada iteración. Adicionalmente, el algoritmo incluye un criterio de convergencia opcional que permite detener el proceso antes de alcanzar el número máximo de iteraciones si las distribuciones de probabilidad convergen suficientemente. La convergencia se verifica calculando la raíz cuadrada de la suma de los cuadrados de las probabilidades para cada elemento y comparándolas con un umbral, en este caso el umbral definido es 0.9.

Al finalizar las iteraciones, la función retorna la mejor solución encontrada, que incluye el centro de cumplimiento, la estación de distribución y el punto final con el menor costo total. Esta solución es la que se considera óptima para la entrega del producto al cliente.

A modo de conclusión, el código proporciona una solución eficiente para la optimización de la cadena de suministro utilizando el algoritmo de Cross-Entropy. A través de la simulación y evaluación iterativa de diferentes combinaciones de centros de cumplimiento, estaciones de distribución y puntos finales, el código identifica la configuración que minimiza el costo total de entrega, asegurando una solución óptima para la logística y distribución de productos. La implementación de estas funciones y el enfoque de optimización permiten una gestión eficaz de los recursos y una reducción significativa de los costos operativos en la cadena de suministro.

# Capítulo 7. DESARROLLO DEL MODELO LLM

En esta sección se detalla la implementación de un modelo basado en BERT para predecir la severidad de incidentes en un entorno de almacenes. Este modelo es capaz de interpretar mensajes de los jefes de los distintos almacenes y centros de procesamiento, clasificando dichos mensajes según la gravedad del incidente descrito. El código del modelo puede encontrarse en el ANEXO D.

El proceso comienza con la carga de un conjunto de datos que contiene descripciones textuales de incidentes y sus correspondientes etiquetas de severidad. Las etiquetas, que siguen un formato específico (por ejemplo, "FIN01 3"), se preprocesan para extraer la severidad numérica. Estos datos han sido generados previamente mediante Chat GPT y han sido procesados y guardados en un Excel.

Una vez cargados los datos, el conjunto de datos se divide en datos de entrenamiento y de prueba, asegurando que el modelo pueda ser evaluado adecuadamente tras el entrenamiento.

El texto de los incidentes se convierte en un formato que el modelo BERT pueda entender a través de un proceso de tokenización. Esta tokenización transforma las frases en secuencias de números (tokens), que son compatibles con el modelo. Se utiliza un tokenizador preentrenado de BERT, garantizando que todas las entradas tengan una longitud uniforme, lo que facilita el procesamiento en lotes.

El modelo BERT preentrenado se ajusta para la tarea de clasificación de severidad. Este modelo ha sido previamente entrenado para comprender y representar el lenguaje, y se especializa para clasificar las entradas en una de las categorías de severidad predefinidas.

Finalmente es hora de entrenar el modelo. Para ello, se definen varios parámetros clave para el entrenamiento, como el número de épocas, el tamaño del lote, y la tasa de aprendizaje. Estos parámetros controlan cómo y durante cuánto tiempo se entrenará el modelo. Además,

se configuran opciones de registro para monitorear el rendimiento del modelo durante el entrenamiento.

El modelo se entrena utilizando el conjunto de datos preparado, y se evalúa su rendimiento con los datos de prueba. Una vez finalizado el entrenamiento, tanto el modelo como el tokenizador se guardan para su uso posterior.

Finalmente, se desarrolla una función para que el modelo entrenado pueda predecir la severidad de incidentes basándose en nuevas descripciones textuales. Esta función toma un texto de entrada, lo procesa a través del modelo y devuelve una predicción de la categoría de severidad.

# Capítulo 8. IMPLEMENTACIÓN DE LA SIMULACIÓN

Una vez se definen los dos modelos, es la hora de testearlos y para ello se ha decidido desarrollar un código para realizar una simulación discreta. Se implementará una simulación donde existirán **26 clientes cada uno de ellos con una identificación de la 'A' a la 'Z'** (sin contar la ñ) y a lo largo de un tiempo irán realizando de manera aleatoria pedidos de un producto y una cantidad determinadas.

En esta sección se explicará como se ha realizado el desarrollo del software encargado de realizar la simulación pertinente.

## *8.1 DESARROLLO DE LA SIMULACIÓN*

El código encontrado en el Anexo C controla el tiempo de simulación en el código principal utilizando la biblioteca `simpy`. El propósito del código es simular el procesamiento y entrega de pedidos, optimizando los recursos disponibles en la cadena de suministro. La simulación se configura para ejecutarse durante un tiempo definido (`SIM_TIME`) y procesar un número específico de pedidos (`NUM_ORDERS`). Finalizará una vez se alcance el SIM_TIME o se hayan cerrado el número de pedidos.

### 8.1.1 CLASE 'ORDER'

Lo primero que se realiza en el código de simulación es la definición de la clase 'Order'. La clase `Order` representa un pedido en la simulación y contiene varios métodos para simular las diferentes etapas del procesamiento y entrega de un pedido. El constructor de la clase inicializa los atributos del pedido, incluyendo el entorno de simulación (`env`), el código del pedido (`order_code`), el cliente, el centro de cumplimiento (`fc`), la estación de distribución (`ds`), el punto final (`end`), los tiempos de procesamiento y transporte, la cantidad y el producto, y el costo predicho.

Dentro de esta clase se definen una serie de métodos que son los encargados de llevar a cabo la simulación del tiempo discreto. Para ello, es necesario usar la función Yield. A continuación, se explican los distintos métodos:

El método `process_fc` simula el procesamiento del pedido en el centro de cumplimiento. Utiliza `yield self.env.timeout(self.fc_time)` para esperar el tiempo de procesamiento correspondiente y actualiza el estado del pedido utilizando `dataWrite.update_order_status`. Similarmente, los métodos `transport_to_ds`, `process_ds`, `transport_to_end`, `process_end` y `transport_to_client` simulan las etapas sucesivas del transporte y procesamiento del pedido desde el centro de cumplimiento hasta la entrega al cliente. Cada método utiliza `yield self.env.timeout()` para simular el tiempo necesario para completar cada etapa y actualiza el estado del pedido en consecuencia.

En caso de que un pedido no sea lo suficientemente grande para satisfacer la condición mínima de transporte, se mantiene en espera hasta que entre otro pedido que se encuentre también por debajo de la capacidad mínima de transporte. Una vez se realiza otro pedido pequeño, estos son procesados como si fuesen uno solo hasta el END Point. Para simular este proceso se utiliza el método `synchronize_with`. Esto es útil para combinar múltiples pedidos pequeños en un solo pedido grande, optimizando así el uso de los recursos.

### 8.1.2 GESTIÓN DE LA SIMULACIÓN

#### *8.1.2.1 Función para gestionar el transporte de cada pedido*

Para realizar la gestión de cada pedido se desarrolla la función `transportation`. Esta función configura el proceso de manejo de un pedido. Inserta un nuevo pedido en la base de datos, actualiza los volúmenes de los centros de cumplimiento y estaciones de distribución, y simula cada etapa del procesamiento y transporte del pedido mediante llamadas a los métodos correspondientes de la clase `Order`. Al completar todas las etapas, el pedido se elimina de la cola de pedidos.

### *8.1.2.2 Función para gestionar los distintos pedidos*

Sin embargo, es la función `place_order` la que crea pedidos aleatorios a diferentes tiempos hasta que finaliza la simulación. Para cada pedido, selecciona un cliente y un producto aleatoriamente, y determina la cantidad del producto basado en una distribución normal. Si la cantidad del pedido es mayor o igual a 25, o si la cola de pedidos pequeños (`small_orders_queue`) está vacía, se llama a la función `cross_entropy_optimization` del modelo para encontrar la mejor combinación de centro de cumplimiento, estación de distribución y punto final. Si la cantidad del pedido es menor a 25, el pedido se añade a la cola de pedidos pequeños. Si la suma de las cantidades en la cola de pedidos pequeños alcanza 25 o más, los pedidos se combinan y se sincronizan sus tiempos con un pedido de referencia.

Los tiempos de procesamiento y transporte se calculan utilizando datos extraídos con el módulo `dataExtraction` y se añaden variaciones aleatorias basadas en una distribución normal para simular retrasos. Los pedidos se crean y se añaden a la cola de pedidos (`order_queue`), y se inician los procesos de transporte y procesamiento correspondientes en el entorno de simulación.

La simulación se ejecuta hasta alcanzar el tiempo máximo de simulación (`SIM_TIME`) o cuando todos los pedidos han sido entregados. Al finalizar, se rellenan los centros de cumplimiento a su capacidad máxima utilizando `dataWrite.refill_FCs`.

En resumen, el código simula el procesamiento y entrega de pedidos en una cadena de suministro utilizando `simpy` para manejar la concurrencia y los eventos en el tiempo. A través de la optimización y sincronización de pedidos, el código busca mejorar la eficiencia en la entrega de productos a los clientes. La estructura modular y la utilización de métodos para cada etapa del proceso permiten una simulación detallada y realista del flujo de pedidos en la cadena de suministro.

# Capítulo 9. ANÁLISIS DE RESULTADOS

En este capítulo se analizarán los resultados del modelo de CE variando los hiperparámetros. Así como los distintos resultados obtenidos en la simulación y por último el entrenamiento del modelo LLM.

## *9.1 MODELO DE CE – ANÁLISIS DE LOS HIPERPARÁMETROS*

Una vez definido y desarrollado el modelo, es hora de determinar cuáles son los hiperparámetros que definen el modelo. Para ello se realizará un análisis de sensibilidad, donde se experimentará para diferentes valores de número de muestras, fracción de élite escogida y número máximo de iteraciones, para obtener soluciones de rápida convergencia y lo más cerca del óptimo.

Para el siguiente caso se examinarán los siguientes valores de los hiperparámetros:

- Número de muestras (n_samples): 10, 20 ,30
- Fracción de élite (elite_frac): 0.3, 0.4, 0.5, 0.7
- Máximo número de iteraciones (max_iters) : 5,10,20

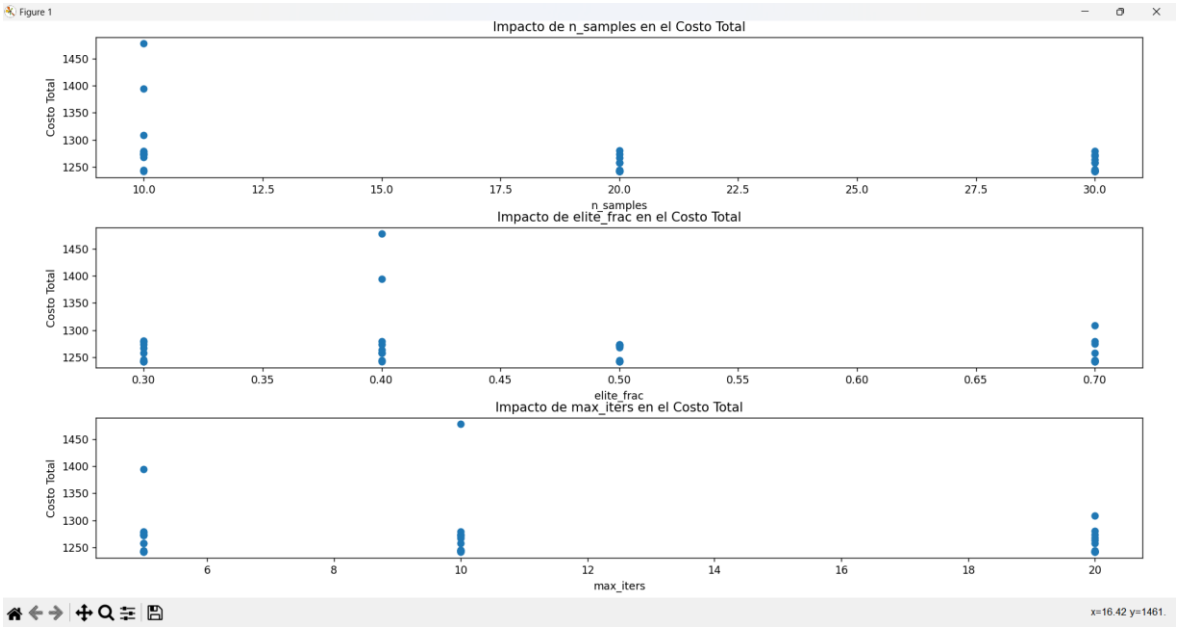



*Ilustración 17. Resultados de análisis de los hiperparámetros*

1. **Impacto de n_samples en el Costo Total:**

Este parámetro controla el número de muestras que se generan en cada iteración del algoritmo. En el gráfico, se observa que el costo total varía significativamente con diferentes valores de n_samples. Para valores bajos de n_samples, el costo total presenta una mayor variabilidad, lo que sugiere que, con pocas muestras, el algoritmo podría no estar capturando adecuadamente la solución óptima.

Al aumentar n_samples, se observa una disminución en la variabilidad, indicando que el algoritmo puede estar convergiendo hacia una solución más estable y posiblemente más cercana al óptimo. Comparando los valores de 20 o 30 muestras, se puede ver como los resultados apenas cambian, por lo tanto, ***es mejor elegir las 20 muestras debido a que el coste computacional es menor.***

2. **Impacto de elite_frac en el Costo Total:**

Este parámetro define la fracción de las mejores muestras que se seleccionan como "elite" para actualizar las distribuciones de probabilidad en cada iteración. El gráfico muestra que con un elite_frac bajo (es decir, seleccionando pocas muestras como elite), el costo total

puede ser más bajo y menos variable. A medida que se incrementa elite_frac, el costo total parece desestabilizarse y aumentarse, lo que indica que un menor número de muestras elite puede guiar mejor la optimización hacia una solución más eficiente.

Para el modelo se seleccionará una ***fracción de élite del 30%.***

3. **Impacto de max_iters en el Costo Total:**

Este parámetro establece el número máximo de iteraciones que el algoritmo ejecutará. Se observa que con pocas iteraciones (max_iters bajo), el costo total es más variable y tiende a ser más alto. A medida que el número de iteraciones aumenta, el costo total disminuye, lo que sugiere que el algoritmo necesita un número suficiente de iteraciones para alcanzar una convergencia efectiva hacia una solución óptima.

En este caso, se ha implementado un early stop en el modelo para que no sea necesario recorrer el número máximo de iteraciones si ya se ha alcanzado un equilibrio.

Por lo tanto, se va a escoger un ***número máximo de iteraciones de 20***, ya que como se aprecia en el modelo la variabilidad es menor.

## *9.2 MODELO LLM*

Tal y como se comentó en el Capítulo 7. , se entrenará el modelo BERT para poder clasificar los mensajes en un rango de gravedad del 1 al 3 siendo 1 un funcionamiento normal, 2 un funcionamiento por debajo de sus máximas capacidades y por último, 3 que indica que el almacén no funciona correctamente. El código utilizado para entrenar al modelo, carga un conjunto de datos que contiene información sobre la severidad de incidentes en formato de texto, los cuales se dividen en datos de entrenamiento y prueba. El modelo BERT es utilizado para la clasificación, siendo entrenado durante tres épocas.

### 9.2.1 ANÁLISIS DE LOS RESULTADOS

Los resultados de cada época, como se observan en la imagen, incluyen varias métricas clave:

```
  warnings.warn(
{'loss': 1.3258, 'grad_norm': 8.273077011108398, 'learning_rate': 1.0000000000000002e-06, 'epoch': 0.13}
{'loss': 1.3046, 'grad_norm': 6.956902027130127, 'learning_rate': 2.0000000000000003e-06, 'epoch': 0.25}
{'loss': 1.3616, 'grad_norm': 6.003896713256836, 'learning_rate': 3e-06, 'epoch': 0.38}
{'loss': 1.2291, 'grad_norm': 9.944498062133789, 'learning_rate': 4.000000000000001e-06, 'epoch': 0.51}
{'loss': 1.1925, 'grad_norm': 9.375588417053223, 'learning_rate': 5e-06, 'epoch': 0.63}
{'loss': 1.1451, 'grad_norm': 9.284863471984863, 'learning_rate': 6e-06, 'epoch': 0.76}
{'loss': 1.0763, 'grad_norm': 8.870650291442871, 'learning_rate': 7.000000000000001e-06, 'epoch': 0.89}
{'eval_loss': 0.9346246123313904, 'eval_runtime': 3.129, 'eval_samples_per_second': 50.176, 'eval_steps_per_second': 6.392,
 'epoch': 1.0}
{'loss': 1.0017, 'grad_norm': 13.047511100769043, 'learning_rate': 8.000000000000001e-06, 'epoch': 1.01}
{'loss': 0.9334, 'grad_norm': 8.021259307861328, 'learning_rate': 9e-06, 'epoch': 1.14}
{'loss': 0.7938, 'grad_norm': 13.41382122039795, 'learning_rate': 1e-05, 'epoch': 1.27}
{'loss': 0.6877, 'grad_norm': 11.488718032836914, 'learning_rate': 1.1000000000000001e-05, 'epoch': 1.39}
{'loss': 0.4696, 'grad_norm': 5.40272855758667, 'learning_rate': 1.2e-05, 'epoch': 1.52}
{'loss': 0.3564, 'grad_norm': 5.199455738067627, 'learning_rate': 1.3000000000000001e-05, 'epoch': 1.65}
{'loss': 0.2221, 'grad_norm': 3.3895838260650635, 'learning_rate': 1.4000000000000001e-05, 'epoch': 1.77}
{'loss': 0.142, 'grad_norm': 2.7830066680908203, 'learning_rate': 1.5e-05, 'epoch': 1.9}
{'eval_loss': 0.04424092546105385, 'eval_runtime': 2.9555, 'eval_samples_per_second': 53.121, 'eval_steps_per_second': 6.76
7, 'epoch': 2.0}
{'loss': 0.0728, 'grad_norm': 0.8314953446388245, 'learning_rate': 1.6000000000000003e-05, 'epoch': 2.03}
{'loss': 0.0365, 'grad_norm': 0.405992716550827, 'learning_rate': 1.7000000000000003e-05, 'epoch': 2.15}
 'epoch': 1.0}
{'loss': 1.0017, 'grad_norm': 13.047511100769043, 'learning_rate': 8.000000000000001e-06, 'epoch': 1.01}
{'loss': 0.9334, 'grad_norm': 8.021259307861328, 'learning_rate': 9e-06, 'epoch': 1.14}
{'loss': 0.7938, 'grad_norm': 13.41382122039795, 'learning_rate': 1e-05, 'epoch': 1.27}
{'loss': 0.6877, 'grad_norm': 11.488718032836914, 'learning_rate': 1.1000000000000001e-05, 'epoch': 1.39}
{'loss': 0.4696, 'grad_norm': 5.40272855758667, 'learning_rate': 1.2e-05, 'epoch': 1.52}
{'loss': 0.3564, 'grad_norm': 5.199455738067627, 'learning_rate': 1.3000000000000001e-05, 'epoch': 1.65}
{'loss': 0.2221, 'grad_norm': 3.3895838260650635, 'learning_rate': 1.4000000000000001e-05, 'epoch': 1.77}
{'loss': 0.142, 'grad_norm': 2.7830066680908203, 'learning_rate': 1.5e-05, 'epoch': 1.9}
{'eval_loss': 0.04424092546105385, 'eval_runtime': 2.9555, 'eval_samples_per_second': 53.121, 'eval_steps_per_second': 6.76
7, 'epoch': 2.0}
```

*Ilustración 18. Resultados del entrenamiento del modelo BERT*

1. **Pérdida (`loss`):** Es una medida de cuánto se desvía la predicción del modelo de las etiquetas reales. La pérdida disminuye de manera constante durante el entrenamiento, comenzando en aproximadamente 1.32 y llegando a 0.0049 hacia el final de la tercera época. Esto sugiere que el modelo está aprendiendo a realizar predicciones más precisas con cada iteración.
2. **Gradiente (`grad_norm`):** Esta métrica monitorea la magnitud de los gradientes durante la retropropagación. Se observa que la norma del gradiente también disminuye a lo largo del entrenamiento, lo que indica que el modelo está convergiendo y ajustando sus pesos de manera precisa.

3. **Tasa de aprendizaje (`learning_rate`):** Comienza en un valor muy bajo, incrementando ligeramente con cada paso de entrenamiento. La tasa de aprendizaje es crucial para asegurar que el modelo aprenda de manera eficiente sin oscilar o estancarse.
4. **Evaluación de la pérdida (`eval_loss`):** Después de cada época, se realiza una evaluación del modelo con el conjunto de prueba. La pérdida de evaluación disminuye de 0.9346 en la primera época a 0.0028 al final de la tercera, lo que indica una mejora constante en el rendimiento del modelo en datos no vistos durante el entrenamiento. Como era de esperar, al usar un modelo ya entrenado, y simplemente adaptarlo a este problema, la precisión es altísima. En caso de haber tenido que desarrollar un modelo propio, se hubieran obtenido con mayor probabilidad peores resultados.

### 9.2.2 CONCLUSIÓN

El entrenamiento del modelo BERT para la clasificación de severidad se desarrolla de manera eficiente, mostrando una clara tendencia de mejora en la precisión con cada época. La reducción constante en la pérdida, tanto durante el entrenamiento como en la evaluación, indica que el modelo está aprendiendo de los datos y generalizando de manera adecuada a datos no vistos. Las métricas relacionadas con los gradientes y la tasa de aprendizaje sugieren que el ajuste de hiperparámetros es adecuado, permitiendo una convergencia sin grandes oscilaciones.

### 9.2.3 COMPORTAMIENTO CON EJEMPLOS

A continuación, se comprobarán los resultados obtenidos para distintos tipos de ejemplos.

1. Caso 1: Se introducirá la frase ‘FIN01 IS DEALING WITH HIGH VOLUMES’ esperando un retorno de una severidad 2, debido a que no se encuentra en sus máximas capacidades. El resultado es el siguiente:

```
Introduce your problem:FIN01 IS DEALING WITH HIGH VOLUMES
The severity score for 'FIN01 IS DEALING WITH HIGH VOLUMES' is Neutral, and the entity is FIN01
The availability of warehouse FIN01 has been modified to 2
```

*Ilustración 19. Prueba 1 del modelo LLM*

Como se aprecia, el resultado es el esperado, y en la base de datos se ha actualizado el valor de FIN01 a 2.

2. Caso 2: Ahora se buscará restaurar el completo funcionamiento de FIN01 con la siguiente frase 'FIN01 is fully available'. Y los resultados son los siguientes:

```
Introduce your problem:FIN01 is fully available
The severity score for 'FIN01 is fully available' is Good, and the entity is FIN01
The availability of warehouse FIN01 has been modified to 1
```

*Ilustración 20. Prueba 2 del modelo LLM*

Tal y como se aprecia, FIN01 ha sido restaurado, y su valor a cambiado a 1 de nuevo tal y como se esperaba.

3. **Caso 3:** En este caso se busca eliminar del sistema un almacén debido a una avería u otro problema. Para ello se usará el almacén W7, con la siguiente frase 'W7 is completaly out of service'.

```
Introduce your problem:W7 is completaly out of service
The severity score for 'W7 is completaly out of service' is Bad, and the entity is W7
The availability of warehouse W7 has been modified to 3
```

*Ilustración 21. Prueba 3 del modelo LLM*

Los resultados muestran como efectivamente el W7 ha sido modificado a un valor 3 que indica que no está disponible y manda sus costes a infinito en el modelo de CE.

4. **Caso 4:** Este es el último caso, y sirve para evitar modificar almacenes que no existan, para ello se leen los almacenes disponibles y se compara con los tokens de la frase introducida. En el caso en el que no haya ningún token que coincida con algún almacén del sistema se obtendrá la siguiente respuesta:

```
Introduce your problem:w7 is completaly out of service
The introduced warehouse is not in the system
```

*Ilustración 22. Prueba 4.1 del modelo LLM*

Como se observa, tiene en cuenta las mayúsculas y en este caso se escribió ‘w7’, en lugar de ‘W7’.

```
Introduce your problem:The Warehouse W7 is back in fully availability
The severity score for 'The Warehouse W7 is back in fully availability' is Good, and the entity is W7
The availability of warehouse W7 has been modified to 1
```

*Ilustración 23. Prueba 4.2 del modelo LLM*

Finalmente, se vuelve a poner a 1 la disponibilidad del almacén.

## *9.3 RESULTADOS DEL MODELO COMPLETO*

En este subapartado, se realizarán una serie de pruebas con el modelo completo para estudiar el comportamiento del sistema.

### 9.3.1 FUNCIONAMIENTO DE LOS MODELOS COMBINADOS

En primer lugar, se realizará una pequeña simulación de cinco pedidos, donde se supondrá que todos los almacenes están completamente disponibles y, por lo tanto, la ruta que se obtendrá será aquella más eficiente en términos temporales y económicos. Una vez obtenida esas rutas óptimas, se modificará la disponibilidad de varios almacenes para poder ver cómo afecta está a la resolución del modelo.

En la siguiente gráfica se muestran los pedidos cuando todos los almacenes están disponibles. Los puntos indican la llegada y salida a un almacén, es decir que, por ejemplo, en la orden 0, el primer punto W6 indica que ha llegado a este almacén y el segundo que ha salido.

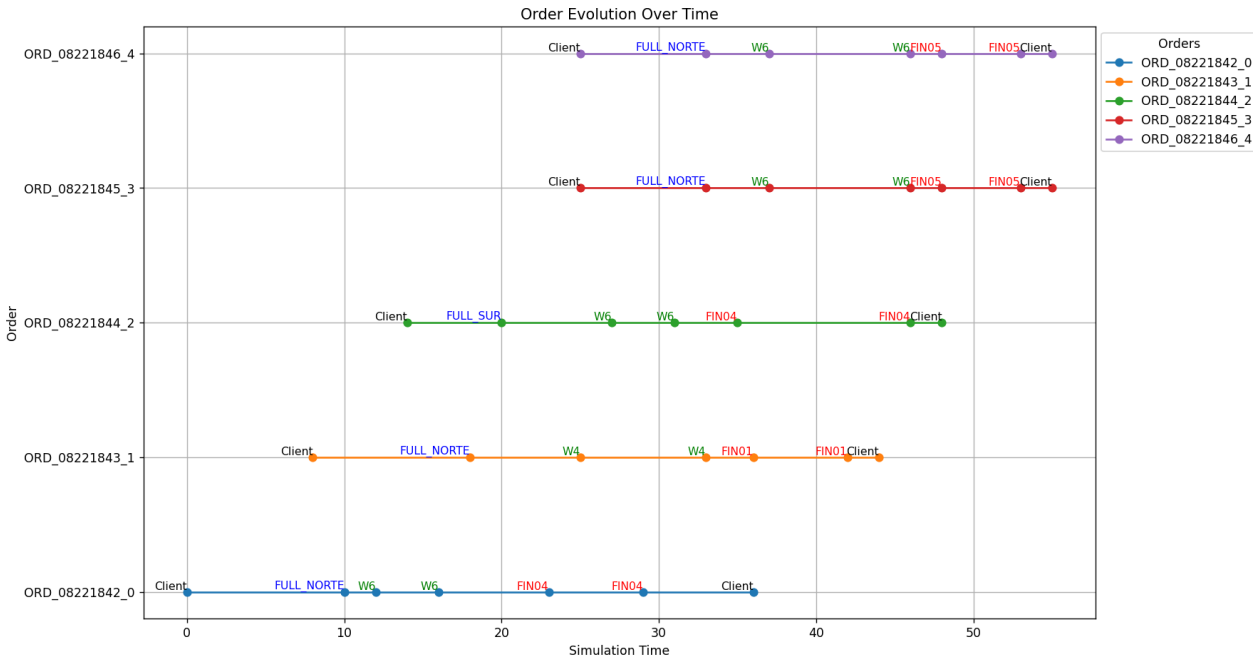



*Ilustración 24. Resultados de la simulación de distintos pedidos*

Como se puede observar, vemos como se envían la gran parte de pedidos por el almacén FULL_NORTE debido a que los costes son menores allí (han sido puestos deliberadamente más bajos para poder probar el correcto funcionamiento del modelo).

A continuación, para los mismos pedidos se procederá a utilizar el modelo LLM para indicar que el almacén FULL_NORTE no está disponible y así podremos ver como este desaparece de la solución inicial.

```
Introduce your problem:FULL_NORTE is no longer available
The severity score for 'FULL_NORTE is no longer available' is Bad, and the entity is FULL_NORTE
The availability of warehouse FULL_NORTE has been modified to 3
```

*Ilustración 25. Bloqueo del almacén FULL_NORTE*

Una vez indicado que el almacén no está disponible, se actualizará automáticamente la base de datos y se cambiará de un 1 a un 3 la columna ‘Availibility’.

| WarehouseID | Type | Subtype | Region | Latitude | Longitude | Availability |
|---|---|---|---|---|---|---|
| FIN10 | End_Point | NULL | Oeste | 40.42341587528859 | -3.695294580116965 | 1 |
| FIN11 | End_Point | NULL | Norte | 40.40354823277607 | -3.6955161710911035 | 1 |
| FIN12 | End_Point | NULL | Sur | 40.41919763063982 | -3.69532264012501 | 1 |
| FULL_ESTE | Fulfillment | NULL | Este | 40.456761 | -3.437198 | 1 |
| FULL_NORTE | Fulfillment | NULL | Norte | 40.521516 | -3.629459 | 3 |
| FULL_OESTE | Fulfillment | NULL | Oeste | 40.415474 | -3.8945 | 1 |
| FULL_SUR | Fulfillment | NULL | Sur | 40.301219 | -3.723882 | 1 |

*Ilustración 26. Cambio de la disponibilidad de FULL_NORTE en la base de datos*

Y estas son las nuevas soluciones:

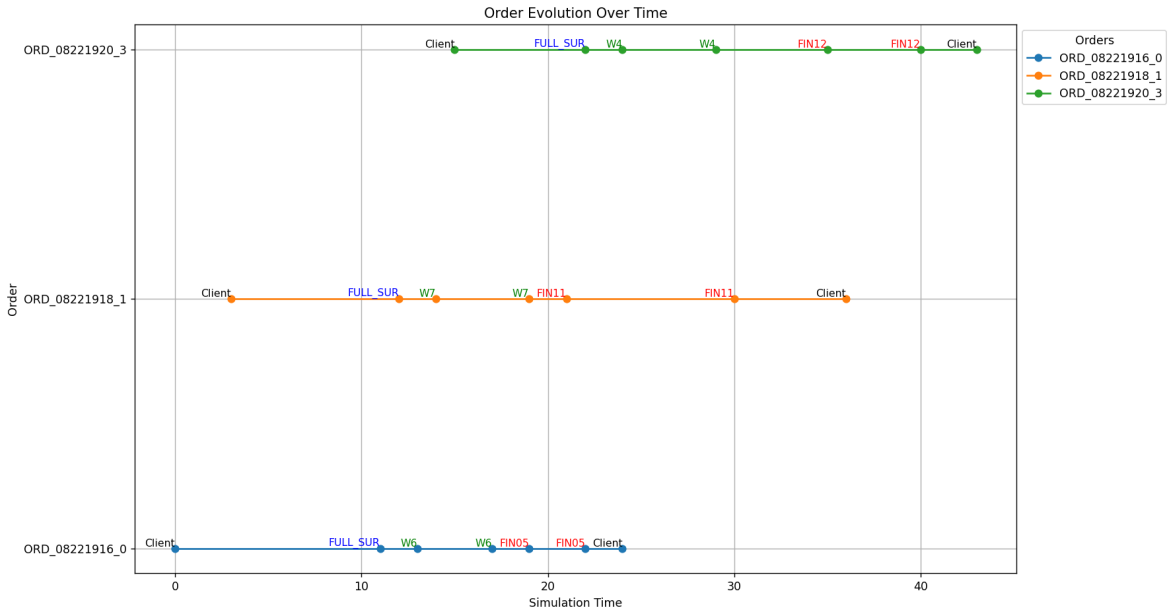



*Ilustración 27. Nuevos resultados sin FULL_NORTE*

Hay que destacar que en esta imagen se ve como ya no hay almacenes FULL_NORTE, ya que este está prohibido usarlo. Ahora destaca el uso del FULL_SUR, lo que se realizará a continuación será poner a este último en una posición intermedia de capacidad para ver como cambia la gráfica.

Notar que en este último gráfico el pedido 2 y 4 no han superado las 25 unidades por lo tanto se han quedado a la espera de nuevos pedidos.

A continuación, actualizamos el nuevo valor de FULL_SUR y procedemos a ver que pasa con la simulación. Esta vez se simularán 20 pedidos.

```
Introduce your problem:FULL_SUR has headcount problems
The severity score for 'FULL_SUR has headcount problems' is Neutral, and the entity is FULL_SUR
The availability of warehouse FULL_SUR has been modified to 2
```

*Ilustración 28. Cambio de la disponibilidad de FULL_SUR en la base de datos*

Y tal y como se puede ver, ahora ya solo obtenemos resultados de los otros dos almacenes restantes, es decir, FULL_ESTE y FULL_OESTE. Aunque en caso de que haya una alta demanda, el almacén FULL_SUR estaría también disponible para su uso limitado. En este caso, una de las cosas a mejorar sería que se actualizase la capacidad máxima del almacén cada vez que se modifica la disponibilidad.


*Ilustración 29. Resultados del modelo tras varios 20 pedidos con FULL_NORTE bloqueado y FULL_SUR restringido.*

Tal y como se puede ver en la Ilustración 31, vemos grandes volúmenes en determinados centros de distribución como puede ser W7 que sale hasta en 7 ocasiones, W3 en 6 ocasiones y W6 hasta en tres ocasiones.

### 9.3.2 PEDIDOS COMPUESTOS

Una de las condiciones interpuestas por el sistema es que las furgonetas tienen que llevar al menos unos pedidos de al menos 25 unidades para ser rentables. En caso de que no se alcancen estos volúmenes, los pedidos quedarán en espera para ser agrupados juntos y entonces ser procesados como un solo pedido. En la siguiente imagen se puede ver un ejemplo.

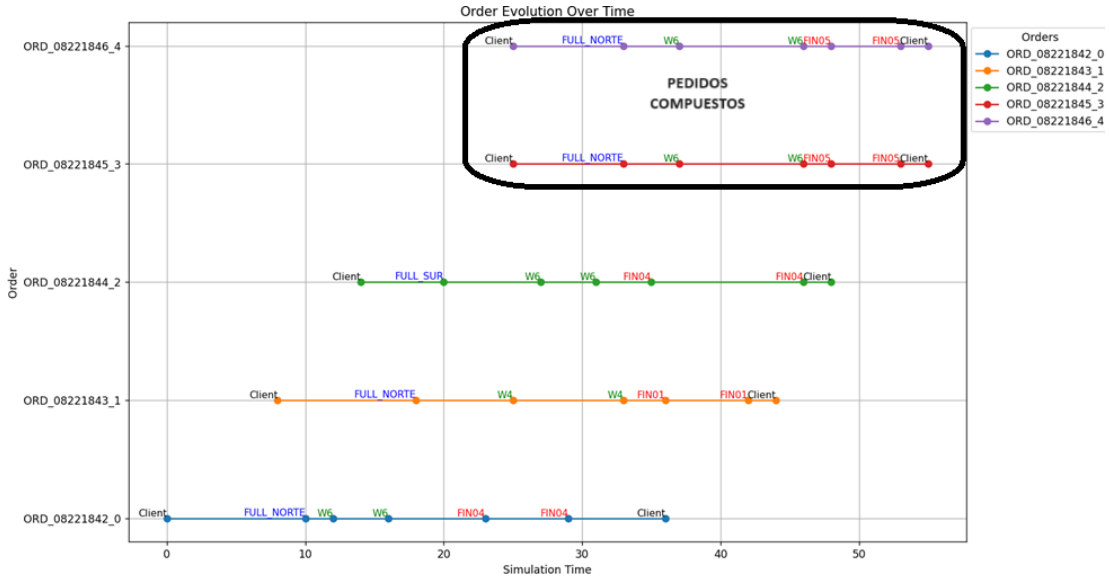



*Ilustración 30. Ejemplo de pedido agrupado*

Como se aprecia en la imagen, los pedidos 3 y 4 (los dos de arriba) han sido agrupados debido a que son pedidos pequeños. El volumen de cada uno de estos pedidos se muestra en la base de datos donde se almacenan los pedidos de los clientes.

| id_pedido | Cliente | FC | DeliveryStation | EndPoint | HoraSalida | HoraLlegada | CostePredicho | TiempoProcesamiento | Cantidad | Producto |
|---|---|---|---|---|---|---|---|---|---|---|
| ORD_08221846_4 | Q | FULL_NORTE | W6 | FIN05 | 33 | 55 | 476.60 | 22 | 11 | Milk |
| ORD_08221845_3 | H | FULL_NORTE | W6 | FIN05 | 33 | 55 | 476.60 | 22 | 17 | Milk |
| ORD_08221844_2 | J | FULL_SUR | W6 | FIN04 | 20 | 48 | 970.44 | 28 | 45 | Cheese |
| ORD_08221843_1 | P | FULL_NORTE | W4 | FIN01 | 18 | 44 | 684.99 | 26 | 26 | Yoghurt |
| ORD_08221842_0 | C | FULL_NORTE | W6 | FIN04 | 10 | 36 | 1199.82 | 26 | 60 | Cheese |

*Ilustración 31. Volúmenes de los pedidos 3 y 4 agrupados*

En la base de datos se aprecia como los dos pedidos son de 11 y 17 por lo tanto se agrupan correctamente y se procesan como uno solo hasta la entrega al cliente final. El modelo solo

tiene en cuenta el primer pedido y el siguiente se añade a este. Es decir, no se recalcula para ver si hay una ruta óptima que deje en un punto intermedio de sus clientes a los dos pedidos.

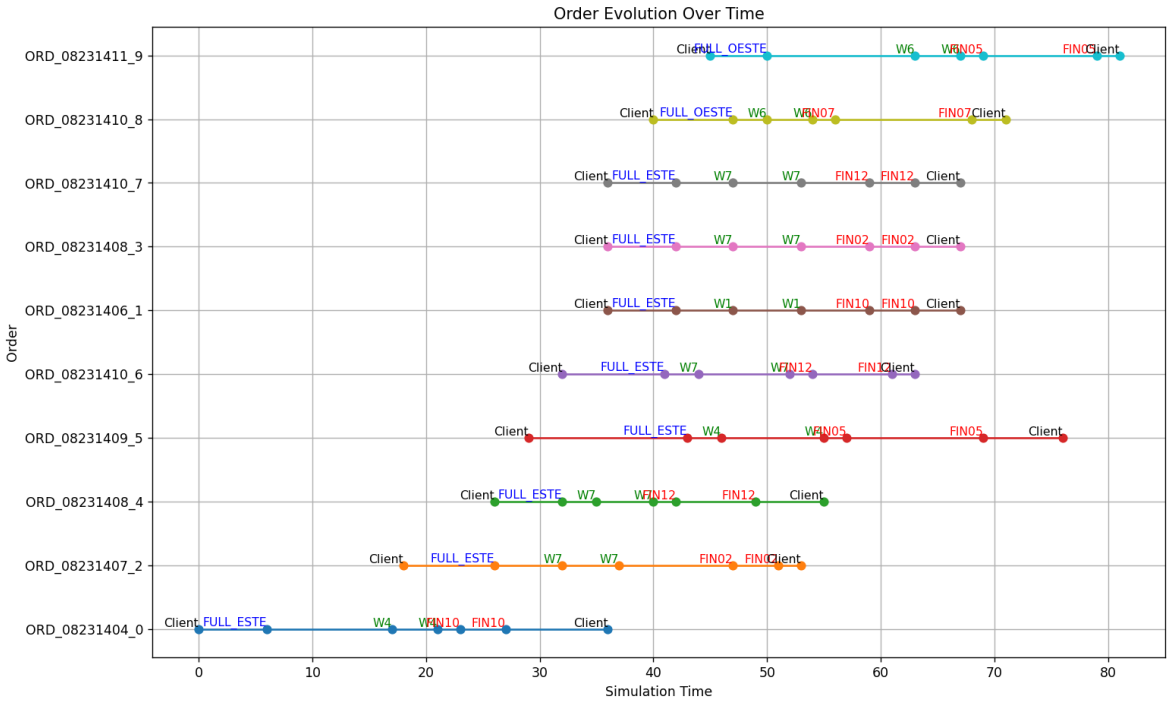
### 9.3.3 GESTIÓN DE LOS INVENTARIOS Y CAPACIDADES

Otra de las restricciones del problema es el número de unidades que se pueden procesar o almacenar en cada centro logístico del modelo. Para ello el código tiene en cuenta la capacidad actual de los almacenes y si estos tienen la posibilidad de procesar el producto.

Para esta gestión se realiza una gestión activa de la capacidad actual en la base de datos, y funciona de la siguiente manera. Cada vez que se realiza un pedido se actualizan las capacidades de todos los almacenes por los que va a transitar el pedido hasta que finalmente sale de ese almacén, es decir si por ejemplo el pedido va por FULL_NORTE, W1 y FIN01 se añadirán los volúmenes a la capacidad actual de cada uno de los tres y no se liberarán hasta que hayan salido de estos. Es decir, se liberará siempre primero después W1 y finalmente FIN01 que es cuando se entrega al cliente. FULL_NORTE al tener el volumen almacenado ya lo único que se hará es restar en el momento en el que se haga el pedido el volumen de este para indicar que se coge de ese almacén.

Ejemplo de funcionamiento: Para este ejemplo se utilizará la hipótesis definida en la introducción donde los almacenes W1 y W4 serán almacenes robóticos con una capacidad muy reducida, en este caso de 50 unidades, pero unos costos significativamente menores que el del resto de centros de distribución, por lo que esperamos que siempre vayan por aquí cuando las unidades totales del pedido sean menores que 50. Para que se vea más claro todavía se va incrementar los gastos del centro de distribución de W1 y así todos los volúmenes serán enviados a W4 si se pueden y sino a otros almacenes.

Estos son los resultados de la simulación de 10 pedidos.




Como se puede observar el primer pedido menor que 50 unidades por lo tanto se irá por el centro de distribución de W4. En este punto, aunque W4 siga siendo más barato, no podrá obtener más volúmenes hasta no se libere otra vez.

```
Order ORD_08231404_0 placed at time 0.00 by client C
DS W4 volumes updated from 0 to 38, where the quantity entering is 38
```

En la gráfica se libera en la hora 21:00 de la simulación, es decir cuando sale de W4.

```
The order ORD_08231404_0 has left the DS: W4 at time 21.00
DS W4 volumes updated from 38 to 0, where the quantity exiting is 38
```

Y es a partir de ese momento cuando el recurso está disponible otra vez. De hecho, se observa como el pedido 4 no entra, pero porque el volumen del pedido es mayor que 50 unidades, concretamente 56.

```
Order ORD_08231408_4 placed at time 26.00 by client Y
DS W7 volumes updated from 96 to 152, where the quantity entering is 56
```

Sin embargo, el pedido 5 si que entra ya que son 36 unidades. Y también se observa como el recurso ya está bloqueado hasta el final de la simulación.

```
Order ORD_08231409_5 placed at time 29.00 by client V
DS W4 volumes updated from 0 to 36, where the quantity entering is 36
```

Y aunque haya pedidos que podrían haber ido por W4, debido a las restricciones impuestas, van por otros centros de distribución. Por ejemplo, la orden 6 que tiene una cantidad de 40:

```
Order ORD_08231410_6 placed at time 32.00 by client D
DS W7 volumes updated from 152 to 192, where the quantity entering is 40
```

Por lo tanto, la restricción funciona correctamente y los almacenes no se ven en ningún caso sobrepasados de capacidad.

*Ilustración 32. Volúmenes de los pedidos 3 y 4 agrupados*

Como se puede ver en esta imagen, los Almacenes W4

```
Order ORD_08231318_1 placed at time 3.00 by client O
DS W4 volumes updated from 0 to 49, where the quantity entering is 49
```

# Capítulo 10. CONCLUSIONES Y TRABAJOS FUTUROS

Tras la implementación inicial de la simulación discreta del proceso de pedidos, se han identificado una serie de pasos futuros necesarios para completar y mejorar el proyecto. Estos pasos incluyen la integración de módulos adicionales, la creación de una interfaz de usuario mediante las cuales los almacenes puedan introducir su problema sin tener que especificar cual es su almacén y la validación de los resultados obtenidos mediante la implementación del modelo en un caso real. A continuación, se describen los futuros desarrollos planeados, así como un resumen de los avances realizados hasta la fecha.

## *10.1 INTEGRACIÓN DE MÓDULOS ADICIONALES*

Uno de los próximos pasos más críticos es la integración de módulos adicionales que permitirán una simulación más detallada y realista. Estos módulos incluirán:

- **Módulo de Rutas de Transporte:** La simulación actual utiliza tiempos de transporte estáticos. Se planea desarrollar un módulo que integre un sistema dinámico de rutas de transporte basado en datos reales de tráfico y disponibilidad de vehículos. Este módulo permitirá la simulación de escenarios más complejos donde el tiempo de entrega varía según las condiciones del entorno.
- **Módulo de Costes de almacén:** Este módulo se utilizará para modelar los costos de procesamiento de cada almacén en función del volumen procesado. Esto permitirá que el sistema tenga una precisión mayor a la hora de determinar el coste que supone enviar una unidad de producto a través de ese almacén. Hasta ahora el coste se supone constante independientemente del volumen procesado.

- **Módulo de predicción de pedidos:** El último y más ambicioso paso sería un modulo que fuera capaz de predecir cuando un cliente va a realizar un pedido, para así ir preparando la red para poder anticiparse a la necesidad del cliente y obtener una mayor velocidad de entrega.

## *10.2 CREACIÓN DE UNA INTERFAZ DE USUARIO AVANZADA*

Lamentablemente, debido a las restricciones temporales que tenía este proyecto, no se ha podido elaborar ni diseñar una interfaz de usuario avanzada para que los mánagers de los almacenes pudieran interactuar con el modelo, sino que se realiza mediante la ejecución de un código de Python. Es por eso que, para continuar con el desarrollo de este proyecto, lo primero que se realizaría es una interfaz adecuada para que los gestores de los almacenes solo tuvieran que describir sus problemas sin necesidad de introducir el nombre de su almacén.

Además, se crearía un dashboard usando PowerBI o Quicksight para poder mostrar la situación en tiempo real de la cadena de suministro. Así como los pedidos que se han realizado, sus rutas asignadas, y los distintitos niveles de inventarios de los almacenes. Esta parte sería clave para identificar posibles problemas del modelo y así mejorarlo todavía más.

## *10.3 IMPLEMENTACIÓN DEL MODELO EN UN CASO REAL*

Antes de implementarlo en un caso real, se hará un estudio de sensibilidad para determinar el grado de adaptación que tiene el modelo en distintos escenarios. Para ello, se crearán múltiples escenarios con diferentes configuraciones de la cadena de suministro, como cambios en la demanda de productos o alteraciones en la red de centros de cumplimiento. Este enfoque permitirá evaluar la capacidad de la simulación para adaptarse a diferentes condiciones y proporcionar soluciones efectivas.

A continuación, es cuando se realizará el diseño y la implementación de un modelo real, el cual usará todo el sistema previamente desarrollado para controlar las rutas y optimizarlas de manera automática y con la capacidad de adaptarse en función de los inputs recibidos por el modelo LLM creado.

## *10.4 ESCALABILIDAD Y MEJORAS EN EL RENDIMIENTO*

Finalmente, para garantizar que el sistema de simulación pueda manejar una mayor escala y complejidad en el futuro, se planean las siguientes mejoras:

1. **Optimización del Código:** Se revisará y optimizará el código para mejorar la eficiencia computacional. Esto incluirá la paralelización de procesos donde sea posible y la utilización de técnicas de reducción de la complejidad computacional.
2. **Escalabilidad en Entornos Distribuidos:** Se investigará la posibilidad de escalar la simulación a entornos distribuidos, utilizando tecnologías como la computación en la nube. Esto permitirá manejar simulaciones de gran escala con miles de clientes y múltiples centros de cumplimiento simultáneamente.

## *10.5 RESUMEN DE LOS AVANCES ACTUALES*

Hasta el momento, se han completado los siguientes desarrollos clave:

- **Desarrollo de la Simulación Básica:** Se ha implementado un prototipo funcional de la simulación discreta utilizando la biblioteca `simpy`. Este prototipo incluye la clase `Order`, que simula el ciclo completo de un pedido, y los métodos para gestionar el transporte y procesamiento de los pedidos.
- **Optimización Inicial de Parámetros:** Se ha implementado una versión inicial del algoritmo de optimización de entropía cruzada para determinar las mejores rutas y centros de cumplimiento para cada pedido.

- **Validación Preliminar:** Se han realizado pruebas preliminares del sistema para asegurar que los resultados sean consistentes y lógicos. Esto ha permitido identificar áreas clave para futuras mejoras y optimizaciones.
- **Desarrollo de la base de datos:** Se ha realizado una primera iteración con toda la estructura y los pipelines necesarios para poder ejecutar el modelo y que los datos se almacenen en una base de datos.
- **Integración del modelo LLM:** Se ha conseguido integrar el modelo utilizado para solventar el problema con un modelo LLM capaz de interpretar el lenguaje natural utilizado por los distintos managers de los almacenes.

En resumen, el proyecto ha alcanzado un estado avanzado, pero aún requiere desarrollos adicionales para lograr una simulación más completa, precisa y eficiente. Los próximos pasos detallados en esta sección garantizarán que la simulación final cumpla con los objetivos planteados y proporcione valor real en la optimización de la cadena de suministro.

# Capítulo 11. ALINEACIÓN CON LOS OBJETIVOS DE DESARROLLO SOSTENIBLE

Este proyecto se caracteriza por cumplir diversos objetivos del desarrollo sostenible, de los cuales destacan tres de ellos:

- **Fin de la pobreza:** Una correcta implementación de este proyecto podría ayudar a disminuir la pobreza. Tal y como se ha podido ver a lo largo de la historia, una mejora de la cadena de suministro permite aumentar el número de productos circulando en el mercado, lo que provoca una caída del precio de estos, significando una mayor accesibilidad a estos productos por parte de la población.
- **Industria, Innovación e Infraestructura:** La alineación de este proyecto con el noveno objetivo de desarrollo sostenible es máxima. Con este sistema, se pretende crear una infraestructura logística resiliente frente a cambios inesperados, promoviendo la industrialización sostenible y claramente fomentando el progreso tecnológico de la sociedad.
- **Acción por el clima:** Finalmente, mediante la implementación de esta herramienta se busca combatir el cambio climático. Esto se conseguirá con la optimización del uso de los recursos disponibles, así como con una mejor gestión de la red de transporte que permita realizar el número máximo de transporte de mercancías con el número mínimo de desplazamientos.

# Capítulo 12. BIBLIOGRAFÍA

[7] Bendezu, K.. "DIAGRAMA UML Y ARQUITECTURA DEL SISTEMA". Sistemas Distribuidos 2013. Febrero, 2013. http://comparape.blogspot.com.es/2013/02/diagrama-uml-y-arquitectura-del-sistema.html.

[8] Herrero Alcántara, T. "Big Data: ¿Moda u oportunidad de negocio para el emprendedor?", Think Big, Octubre 2014. http://blogthinkbig.com/big-data-emprendedor/.

[9] Loeffler, B. "Cloud Computing: What is Infrastructure as a Service", Microsoft Technet Magazine, October 211. https://technet.microsoft.com/en-us/magazine/hh509051.aspx

[10] Vlassis, N.A.; Papakonstantinou, G.; Tsanakas, P. *Dynamic sensory probabilistic maps for mobile robot localization.* Source: Proceedings. 1998 IEEE / RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No.98CH36190) New York, NY, USA: IEEE, 1998.p.718-23 vol.2 of 3 vol. xliv+2010 pp. 11.

[11] S. Ashcroft, "Top 10 AI and ML supply chain companies and solutions," Bizclik Media Ltd, May 24, 2023. Accessed: May 15, 2024. [Online]. Available: https://supplychaindigital.com/digital-supply-chain/top-10-ai-and-ml-supply-chain-solutions

[12] "AWS Supply Chain Command Center for resiliency, visibility, and work orchestration," *Amazon Web Services*, Jul. 10, 2023. https://aws.amazon.com/es/blogs/supply-chain/aws-supply-chain-command-center-for-resiliency-visibility-and-work-orchestration/ (accessed May 19, 2024).

[13] "Optimización de la cadena de suministro — AWS Supply Chain — Amazon Web Services," *Amazon Web Services, Inc.* https://aws.amazon.com/es/aws-supply-chain/ (accessed May 19, 2024).

[14] C. Lamanna, "Introducing the Microsoft Supply Chain Platform, a new approach to designing supply chains for agility, automation and sustainability," *The Official Microsoft Blog*, Nov. 14, 2022. https://blogs.microsoft.com/blog/2022/11/14/introducing-the-microsoft-supply-chain-platform-a-new-approach-to-designing-supply-chains-for-agility-automation-and-sustainability/ (accessed May 19, 2024).

[15] "Digital Supply Chain Solution," *Microsoft Dynamics 365*. https://www.microsoft.com/en-us/dynamics-365/solutions/supply-chain (accessed May 19, 2024).

[16] Maxime C. Cohen & Christopher S. Tang, "The Role of AI in Developing Resilient Supply Chains," Georgetown Journal of International Affairs, Feb. 05, 2024. https://gjia.georgetown.edu/2024/02/05/the-role-of-ai-in-developing-resilient-supply-chains/

[17] A. Upadhyay, "Haversine formula," *Calculate geographic distance on earth*, Feb. 11, 2015. https://www.igismap.com/haversine-formula-calculate-geographic-distance-earth/ (accessed May 28, 2024).

[18] A. Fuentes Penna, "Problema del agente viajero." https://www.uaeh.edu.mx/scige/boletin/tlahuelilpan/n3/e5.html (accessed Jun. 16, 2024).

[19] S. Cook, "Wayback Machine." https://web.archive.org/web/20101212035424/http://www.claymath.org/millennium/P_vs_NP/Official_Problem_Description.pdf

[20] [12]R. Nogales Gine, "Different approaches to Travelling Salesman Problem," Jan. 25, 2022. https://diposit.ub.edu/dspace/bitstream/2445/186678/3/tfg_nogales_gine_roger.pdf (accessed Jun. 16, 2024).

[21] J. Monteiro, "Figure 3.1: Diagram of intersection among classes P, NP, NP-complete...," *ResearchGate*. https://www.researchgate.net/figure/Diagram-of-intersection-among-classes-P-NP-NP-complete-and-NP-hard-problems_fig13_336890186 (accessed Jun. 16, 2024).

[22] B. Fahimnia, H. Davarzani, and A. Eshragh, "Planning of complex supply chains: A performance comparison of three meta-heuristic algorithms," *Computers & Operations Research*, vol. 89, pp. 241–252, doi: 10.1016/j.cor.2015.10.008.

[23] V. Kumar and S. M. Yadav, "A state-of-the-Art review of heuristic and metaheuristic optimization techniques for the management of water resources," *Water Supply*, vol. 22, no. 4, pp. 3702–3728, doi: 10.2166/ws.2022.010.

[24] [Z. Wu, R. H. Möhring, and J. Lai, "Stochastic runtime analysis of a Cross-Entropy algorithm for traveling salesman problems," *Theoretical Computer Science*, vol. 724, pp. 69–86, doi: 10.1016/j.tcs.2017.10.012.

[25] M. Esmaeilikia, B. Fahimnia, J. Sarkis, K. Govindan, A. Kumar, and J. Mo, "A tactical supply chain planning model with multiple flexibility options: an empirical evaluation,"

*Annals of Operations Research*, vol. 244, no. 2, pp. 429–454, Feb. 2014, doi: 10.1007/s10479-013-1513-2.

[26]

# ANEXO A: CÓDIGO FUENTE DEL MODELO

```
import numpy as np
import pandas as pd
import DataExtractionModule as dataExtraction
from transformers import BertTokenizer, BertForSequenceClassification
import torch


def get_beta_value(severity_score):
    if severity_score == 3:
        return float('inf')
    elif severity_score == 1:
        return 1
    elif severity_score == 2:
        return 2
    else:
        return 10  # Handle unexpected severity scores

def calculate_total_cost(fc, ds, end, client, product, quantity):
    """
    This function will calculate the costs of each combination of fd-ds-end-
client proposed by the model.
    Returning the total cost
    """
    fc_stock, max_cap_fc = dataExtraction.extract_stock(fc, product)
    ds_stock, max_cap_ds = dataExtraction.extract_stock(ds, product)
    if fc_stock == 0:
        return float('inf')  # Product is not available at the selected FC
    if (ds_stock + quantity) > max_cap_ds:
        return float("inf") # Product capacity is over

    # Get severity scores and beta values for FC, DS, and END
    severity_fc = dataExtraction.get_severity(fc)
    severity_ds = dataExtraction.get_severity(ds)
    severity_end = dataExtraction.get_severity(end)

    beta_fc = get_beta_value(severity_fc)
    beta_ds = get_beta_value(severity_ds)
    beta_end = get_beta_value(severity_end)

    # print(f"FC: {fc}, Severity: {severity_fc}, Beta: {beta_fc}")
    # print(f"DS: {ds}, Severity: {severity_ds}, Beta: {beta_ds}")
    # print(f"END: {end}, Severity: {severity_end}, Beta: {beta_end}")

    # If any beta value is infinity, return infinity
    if beta_fc == float('inf') or beta_ds == float('inf') or beta_end ==
float('inf'):
        return float('inf')
```

```
    # Use the maximum beta value among FC, DS, and END
    beta = max(beta_fc, beta_ds, beta_end)

    shipping_distance_fc_to_ds =  float(dataExtraction.distance_between_wh(fc,
ds))
    shipping_distance_ds_to_end = float(dataExtraction.distance_between_wh(ds,
end))
    shipping_distance_end_to_client =
float(dataExtraction.distance_to_client(end, client)) # Client has to be a tuple
() with the coordinates
    processing_cost, processing_time = dataExtraction.extract_processing_info(ds,
product)
    gas_price_fc_ds = round(float(dataExtraction.extract_gas_price(fc)[2]), 2)
    gas_price_ds_end = round(float(dataExtraction.extract_gas_price(ds)[2]), 2)
    gas_price_end_client = round(float(dataExtraction.extract_gas_price(end)[2]),
2)
    processing_cost = float(processing_cost)

    # Constant indicating the gas consumption in Litres per km
    consumption = 5  # l/km

    # Computing the component C1 of the objective function, units in Euros
    c_1 = round((shipping_distance_fc_to_ds * gas_price_fc_ds * consumption +
           shipping_distance_ds_to_end * gas_price_ds_end * consumption +
gas_price_end_client*consumption*shipping_distance_end_to_client +
           processing_cost * quantity),5)

    # Avg speed of the trip
    speed = 10  # km/h
    # Computing the component C2 of the objective function, units in hours
    c_2 = round((shipping_distance_fc_to_ds / speed +
           shipping_distance_ds_to_end / speed +
shipping_distance_end_to_client/speed +
           quantity * processing_time),5)

    # Alpha is a constant expressing the cost of time in euros
    alpha = 1  # eur/hour

    total_cost = c_1 + alpha * c_2 * beta
    return round(total_cost, 4)

# Cross-Entropy algorithm for supply chain optimization
def cross_entropy_optimization(fc_list, ds_list, end_list, client_coordinates,
product, quantity, n_samples=20, elite_frac=0.5, max_iters=10):
    """
    This function performs cross-entropy optimization to minimize the total cost
of selecting a fulfillment center (FC),
    distribution station (DS), and endpoint (END) for delivering a product to a
client.

    Parameters:
    - fc_list, ds_list, end_list (list): List of available FCs, DS and End
points.
    - client_coordinates (tuple): Coordinates of the client's location.
```

```
    - product (str): The product to be delivered.
    - quantity (int): The quantity of the product to be delivered.
    - n_samples (int): Number of samples to generate in each iteration (default
is 20).
    - elite_frac (float): Fraction of the top samples considered elite (default
is 0.5).
    - max_iters (int): Maximum number of iterations for the optimization process
(default is 10).

    The function follows these steps:
    1. Initialize probability distributions for selecting FC, DS, and END.
    2. Iteratively sample configurations, evaluate their cost, and update the
probability distributions based on elite samples.
    3. Stop early if the distributions converge sufficiently, indicated by the
square root of the sum of squares of the probabilities being close to 1.
    """
    n_samples = int(n_samples)

    # Initialize probability distributions
    p_fc = np.ones(len(fc_list)) / len(fc_list)
    p_ds = np.ones(len(ds_list)) / len(ds_list)
    p_end = np.ones(len(end_list)) / len(end_list)
    # print("Initial probabilites distribution: ", p_fc)
    best_solution = None
    best_cost = float('inf')
    for iteration in range(max_iters):
        print("Probabilites distribution in itertation:",iteration,":", p_fc)
        print("Probabilites distribution in itertation:",iteration,":", p_ds)
        print("Probabilites distribution in itertation:",iteration,":", p_end)
        # Step 2: Sampling
        samples = []
        for _ in range(n_samples):
            fc = str(np.random.choice(fc_list, p=p_fc))
            ds = str(np.random.choice(ds_list, p=p_ds))
            end = str(np.random.choice(end_list, p=p_end))
            cost = calculate_total_cost(fc, ds, end, client_coordinates, product,
quantity)
            samples.append((fc, ds, end, cost))

        # Step 3: Evaluation and Step 4: Selection
        samples.sort(key=lambda x: x[3])
        n_elite = int(n_samples * elite_frac)
        elite_samples = samples[:n_elite]
        # TO SEE THE SAMPLES CALCULATION
        print(samples)


        # print(f"\nThe elite samples are the following: \n{elite_samples}")
        # Step 5: Updating the probability distributions
        elite_fc = [sample[0] for sample in elite_samples]
        elite_ds = [sample[1] for sample in elite_samples]
        elite_end = [sample[2] for sample in elite_samples]

        p_fc = np.array([elite_fc.count(fc) for fc in fc_list]) / n_elite
```

```
        p_ds = np.array([elite_ds.count(ds) for ds in ds_list]) / n_elite
        p_end = np.array([elite_end.count(end) for end in end_list]) / n_elite

        # Step 6:Update best solution found so far
        if elite_samples[0][3] < best_cost:
            best_solution = elite_samples[0]
            best_cost = elite_samples[0][3]
        # Convergence Check (optional, you can define your own criteria)
        sum_of_squares_fc = np.sum(np.square(p_fc))
        sum_of_squares_ds = np.sum(np.square(p_ds))
        sum_of_squares_end = np.sum(np.square(p_end))

        square_root_fc = np.sqrt(sum_of_squares_fc)
        square_root_ds = np.sqrt(sum_of_squares_ds)
        square_root_end = np.sqrt(sum_of_squares_end)

        # print("Square root for arr_fc:", square_root_fc)
        # print("Square root for arr_ds:", square_root_ds)
        # print("Square root for arr_end:", square_root_end)

        #The early break is done when the distributions of probabilities  square
root of the sum of square is larger than 1
        if square_root_ds > 0.9 and square_root_fc > 0.9 and square_root_end >
0.9:
            print(f"\n Early break done in iteration {iteration}.: Best cost =
{elite_samples[0][3]}")
            break


        print(f"Iteration {iteration}: Best cost = {elite_samples[0][3]}")

    return best_solution



"""
This part of the code was used to test the model without launching the simulation
    - It will oly be executed if THIS file is executed
    - In case we import this file as a module, this code WON'T be executed
"""

simulation = False #Set to false if you dont want to do the model optimization
if __name__ == '__main__':
    import matplotlib.pyplot as plt
    from datetime import datetime
    import string
    # Starting the simulation code
    # stop_thread, simulation_thread = sim.run_simulation()
    #List of FCs, deliver stations and end points
    fc_list = dataExtraction.extract_fc()
    ds_list =dataExtraction.extract_ds()[0]
    ds_subtype = dataExtraction.extract_ds()[1]
    end_list = dataExtraction.extract_end()
    product_list = dataExtraction.extract_product()
```

```
    #  Create the client lists - They will be name with the abcedary letters
    np.random.seed(0)
    client_ids = list(string.ascii_uppercase[:26])
    clients = pd.DataFrame({'Client_ID': client_ids})
    clients['Latitude'] = np.random.uniform(40.405, 40.425, size=26)
    clients['Longitude'] = np.random.uniform(-3.71, -3.695, size=26)

    # Selecting one client
    client = clients.sample(n=1, random_state=2)
    latitude = client.iloc[0]['Latitude']
    longitude = client.iloc[0]['Longitude']
    # Print the random client
    print("\nOrder Client:")
    # print(client.iloc[0]['Client_ID'])
    client_coordinates = (latitude, longitude)


    np.random.seed(0)  # for reproducibility

    #Selecting the end point node making the order
    # end_point = np.random.choice(end_list)
    product_ex = np.random.choice(product_list)
    quantity_ex = int(np.random.normal(40,20))
    # client.iloc[0]
    print("The order was made from:", client.iloc[0]['Client_ID'] ,"of the
product:", product_ex, "quantity = ", quantity_ex)

    if not simulation:
        # Perform optimization
        best_fc, best_ds, best_end, best_cost =
cross_entropy_optimization(fc_list, ds_list, end_list, client_coordinates,
product_ex, quantity_ex)
        print(f"Best FC: {best_fc}, Best DS: {best_ds}, End Point: {best_end},
Cost: {best_cost}")

        # Once obtained the best route is time to write in the data base the new
volumes circulating through the FCs, DS, and End_Point

        order_code = f"ORD_{datetime.now().strftime('%Y%m%d%H%M%S')}"
        delivery_time = datetime.now()


    if simulation:
        # Parámetros a probar
        n_samples_values = [10, 20, 30]
        elite_frac_values = [0.3, 0.4, 0.5, 0.7]
        max_iters_values = [5, 10, 20]

        results = []

        for n_samples in n_samples_values:
            for elite_frac in elite_frac_values:
                for max_iters in max_iters_values:
```

```
                    print(f"Probando con n_samples={n_samples},
elite_frac={elite_frac}, max_iters={max_iters}")
                    best_fc, best_ds, best_end, best_cost =
cross_entropy_optimization(fc_list, ds_list, end_list, client_coordinates,
product_ex, quantity_ex, n_samples=n_samples, elite_frac=elite_frac,
max_iters=max_iters)
                    results.append((n_samples, elite_frac, max_iters, best_cost))

        # Convertir los resultados a un DataFrame para facilitar la visualización
        results_df = pd.DataFrame(results, columns=['n_samples', 'elite_frac',
'max_iters', 'best_cost'])

        # Graficar los resultados
        fig, axs = plt.subplots(3, 1, figsize=(10, 15))

        for i, param in enumerate(['n_samples', 'elite_frac', 'max_iters']):
            axs[i].scatter(results_df[param], results_df['best_cost'])
            axs[i].set_title(f"Impacto de {param} en el Costo Total")
            axs[i].set_xlabel(param)
            axs[i].set_ylabel('Costo Total')

        plt.tight_layout()
        plt.show()

        # Mostrar las mejores combinaciones de parámetros
        best_params = results_df.loc[results_df['best_cost'].idxmin()]
        print(f"Mejores parámetros: n_samples={best_params['n_samples']},
elite_frac={best_params['elite_frac']}, max_iters={best_params['max_iters']},
Costo={best_params['best_cost']}")
```

# ANEXO B: CÓDIGO PARA EXTRAER INFORMACIÓN DE LA BASE DE DATOS

```
"""
    DataExtractionModule.py
    This code is used to define the functions that will extract the info from the
database
"""
import pandas as pd
import mysql.connector
import numpy as np

# print(f"Loading module: {__file__}")

# MySQL connection details
conn_config = {
    'host': 'localhost',
    'user': 'root',
    'password': 'carlos',
    'database': 'DB_TFM_1'
}

# Create MySQL connection
def create_connection():
    conn = mysql.connector.connect(**conn_config)
    return conn

# Function to extract data from the MySQL database
def extract_wh_costs(Warehouse_id, deliver_station, Product):
    conn = create_connection()
    cursor = conn.cursor()

    # Query for the warehouse data
    query = """
    SELECT Product, WarehouseID, Avg_Processing_Cost, Avg_Processing_Time, Stock,
MaxCapacity
    FROM wh_costs_table
    WHERE WarehouseID = %s AND Product = %s
    """

    cursor.execute(query, (Warehouse_id, Product))
    wh_data = cursor.fetchone()

    # Query for the distribution center data
    query = """
    SELECT Product, WarehouseID, Avg_Processing_Cost, Avg_Processing_Time, Stock,
MaxCapacity
```

```
    FROM wh_costs_table
    WHERE WarehouseID = %s AND Product = %s
    """

    cursor.execute(query, (deliver_station, Product))
    ds_data = cursor.fetchone()

    conn.close()

    if wh_data and ds_data:
        return {
            "FC_Avg_processing_cost": wh_data[2],
            "FC_Avg_processing_time": wh_data[3],
            "FC_Stock": wh_data[4],
            "FC_MaxCapacity": wh_data[5],
            "DS_Avg_processing_cost": ds_data[2],
            "DS_Avg_processing_time": ds_data[3],
            "DS_Stock": ds_data[4],
            "DS_MaxCapacity": ds_data[5]
        }
    else:
        return None

def extract_fc():
    conn = create_connection()
    cursor = conn.cursor()

    #Query to extract the WarehousesID for the FCs
    query = """
    SELECT DISTINCT WarehouseID
    FROM warehouses
    WHERE Type = "Fulfillment"
    """

    cursor.execute(query)
    fc_data = cursor.fetchall()

    conn.close()

    if fc_data:
        # print("FC Array returned")
        fc_list = [i[0] for i in fc_data]
        return fc_list
    else:
        print("No data")
        return None

def extract_ds():
    conn = create_connection()
    cursor = conn.cursor()

    #Query to extract the WarehousesID for the FCs
    query = """
    SELECT DISTINCT WarehouseID, subtype
```

```
    FROM warehouses
    WHERE Type = "Distribution"
    """

    cursor.execute(query)
    ds_data = cursor.fetchall()

    conn.close()

    if ds_data:
        # print("DS Array, and subtype array returned")
        fc_list = [i[0] for i in ds_data]
        subtype = [i[1] for i in ds_data]
        return fc_list,subtype
    else:
        print("No data")
        return None


#Example of use
#ds, ds_type = extract_ds()
#print(ds, ds_type)

def extract_end():
    """
    This function extracts from the database the list of End_Points
    """

    conn = create_connection()
    cursor = conn.cursor()

    #Query to extract the WarehousesID for the FCs
    query = """
    SELECT DISTINCT WarehouseID
    FROM warehouses
    WHERE Type = "End_Point"
    """

    cursor.execute(query)
    end_data = cursor.fetchall()

    conn.close()

    if end_data:
        # print("End Points Array returned")
        end_list = [i[0] for i in end_data]
        return end_list
    else:
        print("No data")
        return None

def extract_product():
    """
    This function extracts from the database the list of End_Points
```

```
    """

    conn = create_connection()
    cursor = conn.cursor()

    #Query to extract the WarehousesID for the FCs
    query = """
    SELECT DISTINCT Product
    FROM wh_costs_table
    """

    cursor.execute(query)
    end_data = cursor.fetchall()

    conn.close()

    if end_data:
        # print("End Points Array returned")
        product = [i[0] for i in end_data]
        return product
    else:
        print("No data")
        return None

def distance_between_wh(origin, destination):
    """
    This function extracts from the database the distance between two warehouses.
    The input will be the two wh
    the output will be the distance
    """
    conn = create_connection()
    cursor = conn.cursor()

    #Query to extract the WarehousesID for the FCs
    query = """
    SELECT Distance
    FROM warehouses_relation
    WHERE Origin = %s and Destination = %s
    """

    cursor.execute(query, (origin,destination))
    distance = cursor.fetchone()

    conn.close()

    if distance:
      #  print("Distance between, ", origin,"and destination,", destination," the
two points is:",float(distance[0]))
        return float(distance[0])
    else:
        print("No data")
        return None
#Example of use
# dist = distance_between_wh("FULL_ESTE","W1")
```

```
# print(dist)

def extract_gas_price(warehouse : str):
    """
    This piece pf code is used to extract the gas price of the van used to
transport the products
    The input is the warehouse where we want to extract the data
    The output will be the region, date, and gas price
    """
    conn = create_connection()
    cursor = conn.cursor()

    #Query to extract the WarehousesID for the FCs
    query = """
    SELECT
         g.region
        , date
        , gasprice
    FROM warehouses w left join db_tfm_1.gas_prices g on w.region = g.region
    where 1=1
    AND date = (select max(date) from db_tfm_1.gas_prices)
    AND warehouseid = %s
    ;
    """

    cursor.execute(query, (str(warehouse),))
    price = cursor.fetchone()

    conn.close()

    if price:
        # print("Distance between the two points is:")
        return price[0],price[1],float(price[2])
    else:
        print("No data")
        return None
#Example of use
# result = extract_gas_price("FULL_ESTE")
# if result:
#     region, date, price = result
#     print(f"Region: {region}, Date: {date}, Gas Price: {price}")

def extract_stock(warehouse, product):
    conn = create_connection()
    cursor = conn.cursor()

    #Query to extract the WarehousesID for the FCs
    query = """

    SELECT  Stock, MaxCapacity
    FROM wh_costs_table
    WHERE WarehouseID = %s AND Product = %s
    """
```

```
    cursor.execute(query, (str(warehouse), str(product)))
    stock = cursor.fetchone()


    conn.close()

    if stock:
        #print("Stock:" stock[0]", Max Capacity "{stock[1]})
        return stock[0],stock[1]
    else:
        print("No data")
        return None
# stock, max_cap = extract_stock("W1","Cream")
# print(stock, max_cap)

def extract_processing_info(ds, product):
    conn = create_connection()
    cursor = conn.cursor()

    # Query for the distribution center data
    query = """
    SELECT Avg_Processing_Cost, Avg_Processing_Time
    FROM wh_costs_table
    WHERE WarehouseID = %s AND Product = %s
    """

    cursor.execute(query, (str(ds), str(product)))
    ds_data = cursor.fetchone()

    conn.close()

    return ds_data

# processing_cost, processing_time = extract_processing_info("W1","Cream")
# print(processing_cost, processing_time)
def distance_calculation(lat1, lon1, lat2, lon2):
    """
    Calculate the great circle distance in kilometers between two points
    on the earth (specified in decimal degrees)
    """

    lat1, lon1, lat2, lon2 = map(np.radians, [lat1, lon1, lat2, lon2])

    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = np.sin(dlat/2)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2)**2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1-a))
    r = 6371  # Radius of earth in kilometers. Use 3956 for miles. Determines
return value units.
    return c * r * 1.3

def distance_to_client(end : str, cliente):
    conn = create_connection()
    try:
```

```
        cursor = conn.cursor()

        query = """
            SELECT latitude, longitude
            FROM warehouses
            WHERE WarehouseID = %s
        """

        cursor.execute(query, (end,))
        coordenates = cursor.fetchone()
        if not coordenates:
            raise ValueError("No coordinates found for the specified warehouse
ID.")

        latitude_end, longitude_end = map(float, coordenates)
        latitude_client, longitude_client = map(float, cliente)

        return distance_calculation(latitude_end, longitude_end, latitude_client,
longitude_client)

    except Exception as e:
        print(f"Error: {e}")
        return None
    finally:
        cursor.close()
        conn.close()


def get_severity(wh):
    conn = create_connection()
    try:
        cursor = conn.cursor()

        query = """
            SELECT Availability
            FROM warehouses
            WHERE WarehouseID = %s
        """

        cursor.execute(query, (wh,))
        severity = cursor.fetchone()
        if not severity:
            raise ValueError("No severity found for the specified warehouse ID.")

        return int(severity[0])

    except Exception as e:
        print(f"Error: {e}")
        return None
    finally:
        cursor.close()
        conn.close()
```

```
# client_coords = (40.409762700785464, -3.7037901999999576)  # Example
coordinates
# distance = distance_to_client("W3", client_coords)
# print(f"Distance to client: {distance} km")
"""
    DataExtractionModule.py
    This code is used to define the functions that will extract the info from the
database
"""
import pandas as pd
import mysql.connector
import numpy as np

# print(f"Loading module: {__file__}")

# MySQL connection details
conn_config = {
    'host': 'localhost',
    'user': 'root',
    'password': 'carlos',
    'database': 'DB_TFM_1'
}

# Create MySQL connection
def create_connection():
    conn = mysql.connector.connect(**conn_config)
    return conn

# Function to extract data from the MySQL database
def extract_wh_costs(Warehouse_id, deliver_station, Product):
    conn = create_connection()
    cursor = conn.cursor()

    # Query for the warehouse data
    query = """
    SELECT Product, WarehouseID, Avg_Processing_Cost, Avg_Processing_Time, Stock,
MaxCapacity
    FROM wh_costs_table
    WHERE WarehouseID = %s AND Product = %s
    """

    cursor.execute(query, (Warehouse_id, Product))
    wh_data = cursor.fetchone()

    # Query for the distribution center data
    query = """
    SELECT Product, WarehouseID, Avg_Processing_Cost, Avg_Processing_Time, Stock,
MaxCapacity
    FROM wh_costs_table
    WHERE WarehouseID = %s AND Product = %s
    """

    cursor.execute(query, (deliver_station, Product))
    ds_data = cursor.fetchone()
```

```
    conn.close()

    if wh_data and ds_data:
        return {
            "FC_Avg_processing_cost": wh_data[2],
            "FC_Avg_processing_time": wh_data[3],
            "FC_Stock": wh_data[4],
            "FC_MaxCapacity": wh_data[5],
            "DS_Avg_processing_cost": ds_data[2],
            "DS_Avg_processing_time": ds_data[3],
            "DS_Stock": ds_data[4],
            "DS_MaxCapacity": ds_data[5]
        }
    else:
        return None

def extract_fc():
    conn = create_connection()
    cursor = conn.cursor()

    #Query to extract the WarehousesID for the FCs
    query = """
    SELECT DISTINCT WarehouseID
    FROM warehouses
    WHERE Type = "Fulfillment"
    """

    cursor.execute(query)
    fc_data = cursor.fetchall()

    conn.close()

    if fc_data:
        # print("FC Array returned")
        fc_list = [i[0] for i in fc_data]
        return fc_list
    else:
        print("No data")
        return None

def extract_ds():
    conn = create_connection()
    cursor = conn.cursor()

    #Query to extract the WarehousesID for the FCs
    query = """
    SELECT DISTINCT WarehouseID, subtype
    FROM warehouses
    WHERE Type = "Distribution"
    """

    cursor.execute(query)
    ds_data = cursor.fetchall()
```

```
    conn.close()

    if ds_data:
        # print("DS Array, and subtype array returned")
        fc_list = [i[0] for i in ds_data]
        subtype = [i[1] for i in ds_data]
        return fc_list,subtype
    else:
        print("No data")
        return None


#Example of use
#ds, ds_type = extract_ds()
#print(ds, ds_type)

def extract_end():
    """
    This function extracts from the database the list of End_Points
    """

    conn = create_connection()
    cursor = conn.cursor()

    #Query to extract the WarehousesID for the FCs
    query = """
    SELECT DISTINCT WarehouseID
    FROM warehouses
    WHERE Type = "End_Point"
    """

    cursor.execute(query)
    end_data = cursor.fetchall()

    conn.close()

    if end_data:
        # print("End Points Array returned")
        end_list = [i[0] for i in end_data]
        return end_list
    else:
        print("No data")
        return None

def extract_product():
    """
    This function extracts from the database the list of End_Points
    """

    conn = create_connection()
    cursor = conn.cursor()

    #Query to extract the WarehousesID for the FCs
```

```
    query = """
    SELECT DISTINCT Product
    FROM wh_costs_table
    """

    cursor.execute(query)
    end_data = cursor.fetchall()

    conn.close()

    if end_data:
        # print("End Points Array returned")
        product = [i[0] for i in end_data]
        return product
    else:
        print("No data")
        return None

def distance_between_wh(origin, destination):
    """
    This function extracts from the database the distance between two warehouses.
    The input will be the two wh
    the output will be the distance
    """
    conn = create_connection()
    cursor = conn.cursor()

    #Query to extract the WarehousesID for the FCs
    query = """
    SELECT Distance
    FROM warehouses_relation
    WHERE Origin = %s and Destination = %s
    """

    cursor.execute(query, (origin,destination))
    distance = cursor.fetchone()

    conn.close()

    if distance:
      #  print("Distance between, ", origin,"and destination,", destination," the
two points is:",float(distance[0]))
        return float(distance[0])
    else:
        print("No data")
        return None
#Example of use
# dist = distance_between_wh("FULL_ESTE","W1")
# print(dist)

def extract_gas_price(warehouse : str):
    """
    This piece pf code is used to extract the gas price of the van used to
transport the products
```

```
    The input is the warehouse where we want to extract the data
    The output will be the region, date, and gas price
    """
    conn = create_connection()
    cursor = conn.cursor()

    #Query to extract the WarehousesID for the FCs
    query = """
    SELECT
         g.region
        , date
        , gasprice
    FROM warehouses w left join db_tfm_1.gas_prices g on w.region = g.region
    where 1=1
    AND date = (select max(date) from db_tfm_1.gas_prices)
    AND warehouseid = %s
    ;
    """

    cursor.execute(query, (str(warehouse),))
    price = cursor.fetchone()

    conn.close()

    if price:
        # print("Distance between the two points is:")
        return price[0],price[1],float(price[2])
    else:
        print("No data")
        return None
#Example of use
# result = extract_gas_price("FULL_ESTE")
# if result:
#     region, date, price = result
#     print(f"Region: {region}, Date: {date}, Gas Price: {price}")

def extract_stock(warehouse, product):
    conn = create_connection()
    cursor = conn.cursor()

    #Query to extract the WarehousesID for the FCs
    query = """

    SELECT  Stock, MaxCapacity
    FROM wh_costs_table
    WHERE WarehouseID = %s AND Product = %s
    """

    cursor.execute(query, (str(warehouse), str(product)))
    stock = cursor.fetchone()


    conn.close()
```

```
    if stock:
        #print("Stock:" stock[0]", Max Capacity "{stock[1]})
        return stock[0],stock[1]
    else:
        print("No data")
        return None
# stock, max_cap = extract_stock("W1","Cream")
# print(stock, max_cap)

def extract_processing_info(ds, product):
    conn = create_connection()
    cursor = conn.cursor()

    # Query for the distribution center data
    query = """
    SELECT Avg_Processing_Cost, Avg_Processing_Time
    FROM wh_costs_table
    WHERE WarehouseID = %s AND Product = %s
    """

    cursor.execute(query, (str(ds), str(product)))
    ds_data = cursor.fetchone()

    conn.close()

    return ds_data

# processing_cost, processing_time = extract_processing_info("W1","Cream")
# print(processing_cost, processing_time)
def distance_calculation(lat1, lon1, lat2, lon2):
    """
    Calculate the great circle distance in kilometers between two points
    on the earth (specified in decimal degrees)
    """

    lat1, lon1, lat2, lon2 = map(np.radians, [lat1, lon1, lat2, lon2])

    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = np.sin(dlat/2)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2)**2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1-a))
    r = 6371  # Radius of earth in kilometers. Use 3956 for miles. Determines
return value units.
    return c * r * 1.3

def distance_to_client(end : str, cliente):
    conn = create_connection()
    try:
        cursor = conn.cursor()

        query = """
            SELECT latitude, longitude
            FROM warehouses
            WHERE WarehouseID = %s
```

```
        """

        cursor.execute(query, (end,))
        coordenates = cursor.fetchone()
        if not coordenates:
            raise ValueError("No coordinates found for the specified warehouse
ID.")

        latitude_end, longitude_end = map(float, coordenates)
        latitude_client, longitude_client = map(float, cliente)

        return distance_calculation(latitude_end, longitude_end, latitude_client,
longitude_client)

    except Exception as e:
        print(f"Error: {e}")
        return None
    finally:
        cursor.close()
        conn.close()


def get_severity(wh):
    conn = create_connection()
    try:
        cursor = conn.cursor()

        query = """
            SELECT Availability
            FROM warehouses
            WHERE WarehouseID = %s
        """

        cursor.execute(query, (wh,))
        severity = cursor.fetchone()
        if not severity:
            raise ValueError("No severity found for the specified warehouse ID.")

        return int(severity[0])

    except Exception as e:
        print(f"Error: {e}")
        return None
    finally:
        cursor.close()
        conn.close()


# client_coords = (40.409762700785464, -3.7037901999999576)  # Example
coordinates
# distance = distance_to_client("W3", client_coords)
# print(f"Distance to client: {distance} km")
```

# ANEXO C: CÓDIGO DE LA SIMULACIÓN

```
"""
    This code is used to control the simulation time on the main code. Using
threading we can execute them in parallel
"""
import simpy
import numpy as np
import string
import pandas as pd
import CE_LLM_Model as model
import DataExtractionModule as dataExtraction
import DataWritingModule as dataWrite
from datetime import datetime
import matplotlib.pyplot as plt

# Initialize a list to store the order data
order_data = []

SIM_TIME = 240 #Simulation time until the algorythm stops by its own
NUM_ORDERS = 10 #Select the number of orders that will be placed
#This function is to express as 1h in the simulation all the 0.X h
def round_to_one(x):
    return 1 if x < 1 else x

class Order(object):
    def __init__(self,env, order_code, client, fc, ds, end, fc_time,
transport_fc_ds_time, ds_time,
                 transport_ds_end_time,end_time , transport_end_client_time,
quantity,product, predicted_cost  ) -> None:
        self.env = env
        self.order_code = order_code
        self.client = client
        self.fc = fc
        self.ds = ds
        self.end = end
        self.fc_time = fc_time
        self.transport_fc_ds_time = transport_fc_ds_time
        self.ds_time = ds_time
        self.transport_ds_end_time = transport_ds_end_time
        self.end_time = end_time
        self.transport_end_client_time = transport_end_client_time
        self.quantity = quantity
        self.product = product
        self.predicted_cost = predicted_cost
        self.combined_code = None
        self.synchronized = False
    # def __str__(self):
    #     """ Method defined to print the object"""
```

```
    #     return f"From str method of Test: a is {self.order_code}, b is
{self.client}"
    def process_fc(self):
        yield self.env.timeout(self.fc_time)
        dataWrite.update_order_status(self.order_code, 'HoraSalida',
int(self.env.now))
        self.hora_salida = int(self.env.now)
        print(f"The order {self.order_code} has left the FC: {self.fc} at time
{self.env.now:.2f}")

    def transport_to_ds(self):
        yield self.env.timeout(self.transport_fc_ds_time)
        print(f"The order {self.order_code} has arrived at the DS: {self.ds} at
time {self.env.now:.2f}")

    def process_ds(self):
        yield self.env.timeout(self.ds_time)
        print(f"The order {self.order_code} has left the DS: {self.ds} at time
{self.env.now:.2f}")

    def transport_to_end(self):
        yield self.env.timeout(self.transport_ds_end_time)
        print(f"The order {self.order_code} has arrived at the End Point:
{self.end} at time {self.env.now:.2f}")

    def process_end(self):
        yield self.env.timeout(self.end_time)
        print(f"The order {self.order_code} has left the End Point: {self.end} at
time {self.env.now:.2f}")

    def transport_to_client(self):
        yield self.env.timeout(self.transport_end_client_time)
        dataWrite.update_order_status(self.order_code, 'HoraLlegada',
int(round_to_one(self.env.now)))
        tiempo_procesamiento = -self.hora_salida +
int(round_to_one(self.env.now))
        print(f"EL TIEMPO DE PROCESAMIENTO ES: {tiempo_procesamiento}")
        dataWrite.update_order_status(self.order_code, 'TiempoProcesamiento',
tiempo_procesamiento)
        print(f"The order {self.order_code} has been delivered to the client
{self.client} at time {self.env.now:.2f}")

    def synchronize_with(self, reference_order):
        self.fc_time = reference_order.fc_time
        self.transport_fc_ds_time = reference_order.transport_fc_ds_time
        self.ds_time = reference_order.ds_time
        self.transport_ds_end_time = reference_order.transport_ds_end_time
        self.end_time = reference_order.end_time
        self.transport_end_client_time =
reference_order.transport_end_client_time
        self.synchronized = True

order_data = []
```

```
def transportation(env, order, order_queue):
    """Set ups the process"""
    dataWrite.insert_new_order(order)
    order_data.append({
        'Time': env.now,
        'Order': order.order_code,
        'Status': 'Placed',
        'FC': order.fc,
        'DS': order.ds,
        'End_Point': order.end,
        'Product': order.product,
        'Predicted_Cost': order.predicted_cost
    })

    print(f"Order {order.order_code} placed at time {env.now:.2f} by client
{order.client}")
    dataWrite.update_fc_volumes(order.fc, order.product, order.quantity)
    dataWrite.increase_ds_volume(order.ds, order.product, order.quantity)

    yield env.process(order.process_fc())
    order_data.append({
        'Time': env.now,
        'Order': order.order_code,
        'Status': 'FC Processed',
        'FC': order.fc,
        'DS': order.ds,
        'End_Point': order.end,
        'Product': order.product,
        'Predicted_Cost': order.predicted_cost
    })

    yield env.process(order.transport_to_ds())
    order_data.append({
        'Time': env.now,
        'Order': order.order_code,
        'Status': 'Arrived at DS',
        'FC': order.fc,
        'DS': order.ds,
        'End_Point': order.end,
        'Product': order.product,
        'Predicted_Cost': order.predicted_cost
    })

    yield env.process(order.process_ds())
    order_data.append({
        'Time': env.now,
        'Order': order.order_code,
        'Status': 'DS Processed',
        'FC': order.fc,
        'DS': order.ds,
        'End_Point': order.end,
        'Product': order.product,
        'Predicted_Cost': order.predicted_cost
    })
```

```
    dataWrite.decrease_ds_volume(order.ds, order.product, order.quantity)

    yield env.process(order.transport_to_end())
    order_data.append({
        'Time': env.now,
        'Order': order.order_code,
        'Status': 'Arrived at End Point',
        'FC': order.fc,
        'DS': order.ds,
        'End_Point': order.end,
        'Product': order.product,
        'Predicted_Cost': order.predicted_cost
    })

    yield env.process(order.process_end())
    order_data.append({
        'Time': env.now,
        'Order': order.order_code,
        'Status': 'End Point Processed',
        'FC': order.fc,
        'DS': order.ds,
        'End_Point': order.end,
        'Product': order.product,
        'Predicted_Cost': order.predicted_cost
    })

    yield env.process(order.transport_to_client())
    order_data.append({
        'Time': env.now,
        'Order': order.order_code,
        'Status': 'Delivered',
        'FC': order.fc,
        'DS': order.ds,
        'End_Point': order.end,
        'Product': order.product,
        'Predicted_Cost': order.predicted_cost
    })

    order_queue.remove(order)
    print(f"Order {order.order_code} completed and removed from the queue.")


env = simpy.Environment()
order_queue = []

# Generate random samples for 10 clients
np.random.seed(0)
client_ids = list(string.ascii_uppercase[:26])
clients = pd.DataFrame({'Client_ID': client_ids})
clients['Latitude'] = np.random.uniform(40.405, 40.425, size=26)
clients['Longitude'] = np.random.uniform(-3.71, -3.695, size=26)

fc_list = dataExtraction.extract_fc()
```

```
ds_list =dataExtraction.extract_ds()[0]
ds_subtype = dataExtraction.extract_ds()[1]
end_list = dataExtraction.extract_end()
product_list = dataExtraction.extract_product()


def place_order(env, order_queue, num_orders):
        """
        Creates random deliveries at different times until the simulation ends
        The simulation can end because it has reached the simulation time
        OR
        Because the next loop is finished
        """
        small_orders_queue = []
        last_path = None

        for i in range(num_orders):
            client = clients.sample(n=1).iloc[0]
            latitude = client['Latitude']
            longitude = client['Longitude']
            client_coordinates = (latitude, longitude)
            product = np.random.choice(product_list)
            quantity = max(10,int(np.random.normal(30, 20))) #Max is just to make
sure there are no negative values
            ## quantity = 20

            order_code = f"ORD_{datetime.now().strftime('%m%d%H%M')}_{i}"

            if quantity >= 25 or not small_orders_queue:
                # Call cross_entropy_optimization for orders >= 25 or if
small_orders_queue is empty
                best_fc, best_ds, best_end, best_cost =
model.cross_entropy_optimization(fc_list, ds_list, end_list, client_coordinates,
product, quantity)
                last_path = (best_fc, best_ds, best_end, best_cost)
            else:
                best_fc, best_ds, best_end, best_cost = last_path

            fc = best_fc
            ds = best_ds
            end = best_end
            predicted_cost = round(float(best_cost), 2)

            speed = 10 #km/h

            fc_time =  dataExtraction.extract_processing_info(fc, product)[1] +
int(max(0,np.random.normal(0, 5)))
            transport_fc_ds_time =
int(round_to_one(float(dataExtraction.distance_between_wh(fc, ds))/speed)) +
int(round_to_one(max(0,np.random.normal(0, 5))))
            ds_time = dataExtraction.extract_processing_info(ds, product)[1] +
int(max(0,np.random.normal(0, 5)))
```

```
            transport_ds_end_time =
int(round_to_one(float(dataExtraction.distance_between_wh(ds, end))/speed)) +
int(round_to_one(max(0,np.random.normal(0, 5))))
            end_time = dataExtraction.extract_processing_info(end, product)[1] +
int(max(0,np.random.normal(0, 5)))
            transport_end_client_time =
int(round_to_one(float(dataExtraction.distance_to_client(end,
client_coordinates))/speed)) + int(round_to_one(max(0,np.random.normal(0, 5))))


            order = Order(env, order_code, client['Client_ID'], fc, ds, end,
fc_time, transport_fc_ds_time, ds_time,
                          transport_ds_end_time, end_time,
transport_end_client_time, quantity, product, predicted_cost)

            if quantity < 25:
                small_orders_queue.append(order)
                if sum(o.quantity for o in small_orders_queue) >= 25:
                    combined_order_code =
f"COMB_ORD_{datetime.now().strftime('%m%d%H%M')}_{i}"
                    reference_order = small_orders_queue[1]

                    for o in small_orders_queue:
                        o.combined_code = combined_order_code
                        o.synchronize_with(reference_order)  # Synchronize times
with the reference order
                        order_queue.append(o)
                        env.process(transportation(env, o, order_queue))
                    small_orders_queue.clear()
                    print(f'Orders processed with combined quantity {quantity}')
                else:
                    print(f"Order {order_code} added to small orders queue.
Waiting for more small orders.")
            else:
                order_queue.append(order)
                env.process(transportation(env, order, order_queue))
            # Use a Poisson distribution for the inter-arrival time
            inter_arrival_time = np.random.poisson(5)
            yield env.timeout(inter_arrival_time)
        #Once the process is finished, all the FCs are going to be filled to
their max capcities
        dataWrite.refill_FCs()
env.process(place_order(env, order_queue, NUM_ORDERS))
env.run(until=SIM_TIME)

# Convert order_data to a DataFrame
order_df = pd.DataFrame(order_data)

def create_order_evolution_plot(order_df):
    fig = go.Figure()

    # Add a trace for each unique order
    for order in order_df['Order'].unique():
        order_trace = order_df[order_df['Order'] == order]
```

```
        fig.add_trace(go.Scatter(
            x=order_trace['Time'],
            y=[order] * len(order_trace),
            mode='lines+markers',
            name=order,
            text=[
                f"Status: {row['Status']}<br>"
                f"FC: {row['FC']}<br>"
                f"DS: {row['DS']}<br>"
                f"End Point: {row['End_Point']}<br>"
                f"Product: {row['Product']}<br>"
                f"Predicted Cost: ${row['Predicted_Cost']:.2f}"
                for index, row in order_trace.iterrows()
            ],  # Show the status, FC, DS, End Point, Product, and Cost on hover
            hoverinfo='text',
            line=dict(shape='hv'),  # Horizontal-vertical lines
            marker=dict(size=10)
        ))

    fig.update_layout(
        title="Order Evolution Over Time with Detailed Information",
        xaxis_title="Simulation Time",
        yaxis_title="Orders",
        showlegend=False
    )

    fig.show()
import matplotlib.pyplot as plt

def create_static_order_evolution_plot(order_df):
    plt.figure(figsize=(12, 8))

    # Define a color map for statuses
    status_colors = {
        'FC Processed': 'blue',
        'Arrived at DS': 'green',
        'DS Processed': 'orange',
        'Arrived at End Point': 'red',
        'End Point Processed': 'purple'
    }

    # Plot each order's evolution
    for order in order_df['Order'].unique():
        order_trace = order_df[order_df['Order'] == order]
        times = order_trace['Time']
        statuses = order_trace['Status']
        fcs = order_trace['FC']
        ds = order_trace['DS']
        end_points = order_trace['End_Point']

        # Plot the order's timeline
        plt.plot(times, [order] * len(times), marker='o', linestyle='-',
label=order)
```

```
        # Annotate the points with the FC, DS, or End Point
        for i in range(len(times)):
            if statuses.iloc[i] == 'FC Processed':
                label = fcs.iloc[i]
                color = status_colors['FC Processed']
            elif statuses.iloc[i] in ['Arrived at DS', 'DS Processed']:
                label = ds.iloc[i]
                color = status_colors['Arrived at DS']
            elif statuses.iloc[i] in ['Arrived at End Point', 'End Point
Processed']:
                label = end_points.iloc[i]
                color = status_colors['Arrived at End Point']
            else:
                label = 'Client'
                color = 'black'

            # Avoid label overlapping
            plt.text(times.iloc[i], order, f"{label}", fontsize=9, ha='right',
va='bottom', color=color)

    plt.title('Order Evolution Over Time')
    plt.xlabel('Simulation Time')
    plt.ylabel('Order')
    plt.grid(True)
    plt.legend(loc='upper left', bbox_to_anchor=(1, 1), title='Orders')
    plt.tight_layout()  # Adjust layout to fit labels and legend
    plt.show()

# Run the function to display the static plot
create_static_order_evolution_plot(order_df)
```

# ANEXO D: CÓDIGO DEL MODELO LLM

```
import pandas as pd
from sklearn.model_selection import train_test_split
from transformers import BertTokenizer
import torch
from transformers import BertForSequenceClassification, Trainer,
TrainingArguments

# Load the dataset
data = pd.read_csv('warehouse_severity_dataset.csv')

# Extract the severity label from the text format "FIN01 3"
data['severity'] = data['label'].apply(lambda x: int(x.split()[1]))

# Split the data into training and testing sets
train_data, test_data = train_test_split(data, test_size=0.2)

# Initialize the tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Tokenize the sentences
train_encodings = tokenizer(train_data['text'].tolist(), truncation=True,
padding=True)
test_encodings = tokenizer(test_data['text'].tolist(), truncation=True,
padding=True)

# Create a custom dataset class for PyTorch
class SeverityDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in
self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx], dtype=torch.long)
        return item

    def __len__(self):
        return len(self.labels)

# Create the dataset objects
train_dataset = SeverityDataset(train_encodings, train_data['severity'].tolist())
test_dataset = SeverityDataset(test_encodings, test_data['severity'].tolist())

# Load the BERT model for sequence classification
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',
num_labels=4)
```

```
# Define training arguments
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
    evaluation_strategy="epoch",  # Evaluate after each epoch
    save_strategy="epoch",  # Save model after each epoch
    report_to='tensorboard',  # Save logs for TensorBoard
)

# Initialize the Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
)

# Train the model
trainer.train()

# Save the trained model and tokenizer
model.save_pretrained('./llm_trained_model')
tokenizer.save_pretrained('./llm_trained_model')

# Evaluate the model and print the evaluation results
eval_result = trainer.evaluate()
print(f"Evaluation result: {eval_result}")

# Function to make predictions on new text
def predict(text):
    encoding = tokenizer(text, return_tensors='pt', truncation=True,
padding=True)
    outputs = model(**encoding)
    logits = outputs.logits
    predicted_class = torch.argmax(logits, dim=1).item()
    return predicted_class

# Example usage
sentence = "FIN05 is broken"
severity_score = predict(sentence)
print(f"The severity score for '{sentence}' is {severity_score}")
```

# ANEXO E: CÓDIGO PARA EJECUTAR EL MODELO LLM

```
from transformers import BertTokenizer, BertForSequenceClassification
import torch
import DataExtractionModule as dataExtraction
import DataWritingModule as dataWriting

# Load model and tokenizer
model = BertForSequenceClassification.from_pretrained('./llm_trained_model')
tokenizer = BertTokenizer.from_pretrained('./llm_trained_model')

def tokenize_input(text):
    return tokenizer(text, return_tensors='pt', truncation=True, padding=True,
max_length=512)

def predict(text):
    inputs = tokenize_input(text)
    with torch.no_grad():
        outputs = model(**inputs)
    logits = outputs.logits
    predicted_class = torch.argmax(logits, dim=1).item()
    return predicted_class

def get_severity_label(prediction):
    severity_mapping = {0: 'Very Good', 1: 'Good', 2: 'Neutral', 3: 'Bad'}
    return severity_mapping.get(prediction, 'Unknown')
def extract_entity_from_phrase(phrase):

    fc_list = dataExtraction.extract_fc()
    ds_list =dataExtraction.extract_ds()[0]
    end_list = dataExtraction.extract_end()
    entity_set = set(fc_list + ds_list + end_list)
    words = set(phrase.split())
    for word in words:
        if word in entity_set:
            return word

#  User interface:
sentence = input("Introduce your problem:")
severity_score = predict(sentence)
severity_label = get_severity_label(severity_score)
entity = extract_entity_from_phrase(sentence)
if entity:
    print(f"The severity score for '{sentence}' is {severity_label}, and the
entity is {entity}")
    dataWriting.update_severity(entity,severity_score)
else:
    print(f"The introduced warehouse is not in the system ")
```

# ANEXO E: CÓDIGO CON MÓDULOS PARA ESCRIBIR EN LA BASE DE DATOS

```
"""
    DataWritingModule.py
    This file is used to define all the writing functions to update the DB with
new volumes
"""

import mysql.connector
import DataExtractionModule as dataExtraction
from mysql.connector import Error
# MySQL connection details
conn_config = {
    'host': 'localhost',
    'user': 'root',
    'password': 'carlos',
    'database': 'DB_TFM_1'
}

# Create MySQL connection
def create_connection():
    conn = mysql.connector.connect(**conn_config)
    return conn

def update_volumes(fc, ds, product, quantity):
    fc_quantity = dataExtraction.extract_stock(fc,product) - quantity

def insert_new_order(order):
    """
    This function is in charge of inserting the new orders placed in the
simulation
    """
    conn = create_connection()
    try:
        cursor = conn.cursor()
        sql_insert_query = """
        INSERT INTO trayecto (id_pedido, Cliente, FC, DeliveryStation, EndPoint,
Cantidad, Producto, CostePredicho, IDPedidoCombinado)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s,%s)
        """
        order_data = (
            str(order.order_code),
            str(order.client),
            str(order.fc),
            str(order.ds),
            str(order.end),
```

```
            int(order.quantity),
            str(order.product),
            round(float(order.predicted_cost),2),
            str(order.combined_code)
        )
        cursor.execute(sql_insert_query, order_data)
        conn.commit()
    except Error as e:
        print(f"Error in the connection: {e}")

def update_order_status( order_code, column, value):
    """
    This function is used to update the columns 'HoraSalida', 'HoraLlegada' and
'TiempoProcesamiento'
    everytime an order lefts the FC or is delivered to the customer
    """
    conn = create_connection()
    try:
        cursor = conn.cursor()
        sql_update_query = f"UPDATE trayecto SET {column} = %s WHERE id_pedido =
%s"
        cursor.execute(sql_update_query, (value, order_code))
        conn.commit()
    except Error as e:
        print(f"Error: {e}")

def update_fc_volumes(wh_id,product, quantity):
    """
    Updates the FC volumes everytime a new order is placed
    """
    previous_stock,_= dataExtraction.extract_stock(wh_id,product)
    stock = int(previous_stock)-quantity

    conn = create_connection()
    try:
        cursor = conn.cursor()
        sql_update_query = f"UPDATE wh_costs_table SET Stock = %s WHERE
WarehouseID = %s and Product = %s"
        cursor.execute(sql_update_query, (stock,str(wh_id), str(product)))
        conn.commit()
        # print(f"FC {wh_id} volumes updated from {previous_stock} to {stock},
where the quantity ordered is {quantity} ")
    except Error as e:
        print(f"Error while trying to update FC vols: ", e)

# update_fc_volumes("FULL_NORTE","Cheese",20)

def increase_ds_volume(ds_id, product, quantity):
    """
    Updates the DS volumes blocking its capacity once an order is placed
    (the capacity is added before the order enters the DS).
    """
    previous_stock,_= dataExtraction.extract_stock(ds_id,product)
    stock = int(previous_stock)+quantity
```

```
    conn = create_connection()
    try:
        cursor = conn.cursor()
        sql_update_query = f"UPDATE wh_costs_table SET Stock = %s WHERE
WarehouseID = %s and Product = %s"
        cursor.execute(sql_update_query, (stock,str(ds_id), str(product)))
        conn.commit()
        print(f"DS {ds_id} volumes updated from {previous_stock} to {stock},
where the quantity entering is {quantity} ")
    except Error as e:
        print(f"Error while trying to increase DS vols: ", e)

def decrease_ds_volume(ds_id, product, quantity):
    """
    This function updates the DS volumes once the order lefts the DS.
    """
    previous_stock,_= dataExtraction.extract_stock(ds_id,product)
    stock = int(previous_stock) - quantity
    conn = create_connection()
    try:
        cursor = conn.cursor()
        sql_update_query = f"UPDATE wh_costs_table SET Stock = %s WHERE
WarehouseID = %s and Product = %s"
        cursor.execute(sql_update_query, (stock,str(ds_id), str(product)))
        conn.commit()
        print(f"DS {ds_id} volumes updated from {previous_stock} to {stock},
where the quantity exiting is {quantity} ")
    except Error as e:
        print(f"Error while trying to decrease DS vols: ", e)

def refill_FCs():
    """
   This function is in charge of filling the FCs at their maximum storaging
capacity
   , once the simulation has ended.
    """
    conn = create_connection()
    try:
        cursor = conn.cursor()
        sql_update_query = "UPDATE wh_costs_table SET Stock = MaxCapacity where
WarehouseID like ('FULL_%')"
        cursor.execute(sql_update_query)
        conn.commit()
        print(f"FCs has been resupplied and are at max inventary")
    except Error as e:
        print(f"Error while trying to refill FCs: ", e)

def initialize_DSs():
    """
   This function is in charge of filling the FCs at their maximum storaging
capacity
   , once the simulation has ended.
    """
```

```
    conn = create_connection()
    try:
        cursor = conn.cursor()
        sql_update_query = "UPDATE wh_costs_table SET Stock = 0 where WarehouseID
like ('W%')"
        cursor.execute(sql_update_query)
        conn.commit()
        print(f"DSs has been initialized to 0")
    except Error as e:
        print(f"Error while trying to initialize DS vols: ", e)


if __name__ == '__main__':
    initialize_DSs()
    refill_FCs()

def update_severity(wh, new_severity):
    """
   This function is in charge of updating the availability of the different
warehouses
    """
    conn = create_connection()
    try:
        cursor = conn.cursor()
        sql_update_query = "UPDATE warehouses SET Availability = %s where
WarehouseID = %s"
        cursor.execute(sql_update_query,(new_severity,wh))
        conn.commit()
        print(f"The availability of warehouse {wh} has been modified to
{new_severity}")
    except Error as e:
        print(f"Error while trying to update the availability: ", e)
```