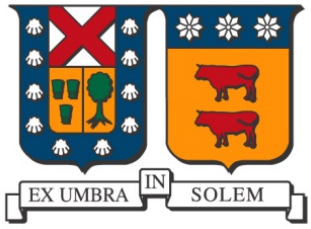
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE



“MODELO DE PREDICCIÓN DE CONSUMO DE RECURSOS COMPUTACIONALES”

FRANCO ALEJANDRO CABEZAS POBLETE

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Roberto Asín

Agosto - 2024

DEDICATORIA

Para la Kenita, si se pudo

AGRADECIMIENTOS

Quisiera agradecer a mi madre Carolyn, a mis tías Leticia y Margarita y a mi primo Juan, quienes desde que tengo memoria me han apoyado en todo. Se han preocupado por mi bienestar y se han sacrificado para que pueda tener un buen futuro. Muchas gracias por la confianza depositada en mí, al sostener mi día a día al partir a Valparaíso. También me gustaría agradecer a mis tíos Luchines, la familia Contreras, la familia Yáñez y mis amigos por el cariño incondicional que he recibido a lo largo de mi vida.

Un agradecimiento especial a mi profesor guía, Roberto Asín. Sin su constante preocupación y apoyo, esta memoria no hubiera sido posible. Muchas gracias por ser un docente de excelente calidad, por su pasión por la investigación y por siempre estar disponible en caso de cualquier duda u consulta que me surgiera. Ha sido un verdadero gusto trabajar con un profesional de tal calibre.

A Paulina Vega, le agradezco por ser un pilar fundamental en mi vida universitaria, ser una gran inspiración en momentos de estrés, gracias por siempre ponerme los pies en la tierra, por soportarme, por crecer conmigo y por acompañarme a lo largo de estos cinco años.

Además, quiero agradecer a Angelo Miranda que en más de alguna vez moleste en altas horas de la noche para resolver dudas y que sin su ayuda hubiera sido aún más difícil la realización de este trabajo.

Me gustaría agradecer al equipo de *FurIA*, con el que he estado trabajando los últimos años, por su gran paciencia, esperando que finalice este proceso y por tener confianza en mí como profesional. Muchas gracias por darme un entorno genial de trabajo.

Sin dudas conocí a increíbles personas a lo largo de mi vida universitaria, por lo que quiero agradecerles a todos, ya qué de alguna forma aportaron su granito de arena para que lograse completar esta meta en mi vida. De verdad, muchas gracias.

RESUMEN

Resumen— Este trabajo aborda la problemática de **predecir el consumo de recursos computacionales en una máquina objetivo a partir del consumo de recursos y características una máquina base**. En específico, el objetivo es predecir una de las métricas principales, **el tiempo total de ejecución de un programa en la máquina objetivo**, mediante la creación de un conjunto de datos diverso y representativo y la implementación de cuatro modelos de aprendizaje automático. Al compararlos con un estudio del estado del arte, se obtuvo un desempeño competitivo, donde el mejor modelo fue XGBoost, y la métrica utilizada para compararlos fue el *Mean Absolute Percentage Error* (MAPE).

Palabras Clave— Aprendizaje automático; Tiempo de ejecución; Redes neuronales; Análisis de desempeño; Predicción de recursos computacionales

ABSTRACT

***Abstract*—** This work addresses the problem of **predicting the consumption of computational resources on a target machine based on the resource consumption and characteristics of a base machine**. Specifically, the objective is to predict one of the main metrics, **the total execution time of a program on the target machine**, by creating a diverse and representative dataset and implementing four machine learning models. When compared to a state-of-the-art study, a competitive performance was achieved, where the best model was XGBoost, and the metric used for comparison was the Mean Absolute Percentage Error (MAPE).

***Keywords*—** Machine Learning; Execution Time; Neural Networks; Performance Analysis; Computational Resource Prediction

GLOSARIO

ACET: Average-Case Execution Time.
ANN: Artificial Neural Networks.
CPU: Central Process Unit.
DI: Departamento de Informática.
EDA: Exploratory Data Analysis.
FLNN: Functional Link Neural Network.
HPC: High-Performance Computing.
IOPS: Input/Ouput Operations Per Second.
KNN: K-Nearest Neighbor.
LSTM: Long Short-Term Memory.
MAE: Mean Absolute Error.
MAPE: Mean Absolute Percentage Error.
ML: Machine Learning.
MSE: Mean Squared Error.
OS: Operating System.
PQR: Predicting Query Runtime.
PQR2: Predicting Query Runtime Regression.
R^2: Coeficiente de determinación.
RAM: Random Access Memory.
RMSE: Root Mean Square Deviation.
RNN: Recurrent Neural Networks.
SHAP: SHapley Additive exPlanations.
SVM: Support Vector Machine.
TFT: Temporal Fusion Transformer.
TSP: Travelling Salesman Problem.
KNP: Knapsack Problem.
GRU: Gated Recurrent Units.
UTFSM: Universidad Técnica Federico Santa María.
VM: Virtual Machine.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

En los entornos como la computación de alto desempeño, sistemas auto-escalables, *data-centers* y computación en grilla, la capacidad de predecir con precisión el consumo de recursos computacionales se ha vuelto una necesidad crítica. Con el avance en la complejidad de los algoritmos y la diversidad de arquitecturas de *hardware*, optimizar la asignación de recursos, como el tiempo de ejecución, memoria de acceso aleatorio (RAM) utilizada, Operaciones de entrada y salida por segundo (IOPS), energía consumida, entre otros, es esencial evitar para sobrecargas y reducir costos operativos. Esta memoria nace de este contexto, abordando la problemática de **predecir el consumo de recursos computacionales en una máquina objetivo a partir del consumo de recursos y características en una máquina base**, en específico el objetivo será predecir una de las métricas principales, **el tiempo total de ejecución de un programa en la máquina objetivo**.

Además, se aborda la falta de modelos robustos y precisos que puedan predecir el consumo de recursos en diferentes configuraciones de *hardware*. La propuesta de solución implica la construcción de cuatro modelos de *Machine Learning* (ML) que, entrenados con datos representativos y diversos, sean capaces de ofrecer predicciones precisas sobre el tiempo de ejecución en una máquina distinta a la que se realizaron las mediciones iniciales.

Para realizar esta propuesta, se obtuvieron datos desde un conjunto diverso de computadores, incluyendo el *cluster* de computación de alto desempeño *Savio*, facilitado por la Universidad de California, Berkeley, y otro conjunto con computadores de escritorio y *notebooks* personales. Posteriormente, se realizó el procesado de los datos con el objetivo de crear un conjunto de datos, así como la implementación de cuatro modelos de ML. Los resultados obtenidos se presentan y analizan en profundidad, comparándolos con un modelo del estado del arte. Finalmente, se discuten las conclusiones generales, el cumplimiento de los objetivos planteados y se proponen líneas de trabajo futuro que podrían ampliar el alcance de los resultados obtenidos en esta memoria.

CAPÍTULO 1
DEFINICIÓN DEL PROBLEMA

En este capítulo se definirá el contexto donde se enmarca el problema. Luego se establecerán tanto el objetivo general y los objetivos específicos del estudio. Por último, se analizará el impacto inicial que conllevaría solucionar el problema definido.

1.1. Contexto

En entornos como la investigación algorítmica, la optimización de infraestructuras, los sistemas auto-escalables, *datacenters* y computación en grilla se enfrenta a la creciente necesidad de predecir y gestionar eficientemente los recursos computacionales como el tiempo de ejecución, RAM utilizada, IOPS, energía consumida, uso de disco, entre otros; estos recursos interactúan de diversas maneras con la complejidad del algoritmo evaluado y con las características de las máquinas objetivo y base.

La capacidad de prever con precisión el consumo de estos recursos para diferentes algoritmos es esencial para evitar sobrecargas, optimizar el rendimiento y garantizar la eficiencia en términos de costos y energía. A pesar de que se han propuesto soluciones en el pasado, éstas suelen estar basadas en modelos tradicionales y “hacen suposiciones simplistas sobre las cargas de trabajo y estadísticas del sistema” [Schmidt *et al.*, 2018], pasando por alto la complejidad de las arquitecturas de aprendizaje de máquina modernas. Dentro de este contexto, surge la necesidad de desarrollar un modelo robusto que, aprovechando las técnicas más actuales de aprendizaje automático y tomando en cuenta las particularidades y diversas configuraciones de los computadores modernos, ofrezca predicciones precisas sobre el consumo de recursos computacionales.

1.2. Objetivos

Se presentan los siguientes objetivos a alcanzar con el desarrollo del estudio:

1.2.1. Objetivo general

Construir un modelo de ML que prediga el consumo de recursos computacionales en una máquina objetivo a partir del consumo de recursos y características una máquina base.

1.2.2. Objetivos específicos

O1) Elaborar un conjunto de datos diverso y representativo para entrenar el modelo de ML, basado en las estadísticas del consumo de recursos de varios algoritmos en diferentes máquinas.

O2) Evaluar la calidad del modelo de ML aplicando métricas estándar (MSE, MAE, R^2).

O3) Comparar los resultados de la solución propuesta con los resultados del estado del arte.

O4) Comunicar a la comunidad académica los resultados de esta investigación.

1.3. Impacto inicial de solucionar el problema

Los Investigadores, administradores de sistemas y empresas se verían beneficiados por este modelo dado que su trabajo depende, en gran medida, de la correcta asignación y uso de recursos computacionales al momento de ejecutar diversos algoritmos y aplicaciones, pues una asignación inadecuada o uso de un equipo no adecuado para la ejecución del algoritmo/aplicación basada en predicciones erróneas puede tener repercusiones operativas.

Factores tecnológicos, como el avance en *hardware* y la diversidad de arquitecturas, complican realizar una predicción precisa. Además, comparar el uso de recursos de los algoritmos en cada equipo es caro y dificultoso, ya que se requeriría volver a ejecutar muchos y largos experimentos. Por otra parte, en el contexto organizacional, el costo asociado a la sobrecarga o subutilización de recursos, tienen un papel importante porque muchas empresas e instituciones de investigación se enfrentan a desafíos al tratar de optimizar los recursos de los que disponen. Resolver este problema tendría un impacto significativo en la optimización del uso de recursos, ya que una predicción precisa en diferentes arquitecturas permitiría una planificación adecuada, evitando costos innecesarios y maximizando la utilización de recursos, además del tiempo ahorrado en la comparación de algoritmos entre máquinas, al no tener que ejecutarlos para realizar la comparación. La solución podría ser trascendental, especialmente considerando la creciente demanda de recursos computacionales en diversas aplicaciones.

CAPÍTULO 2
MARCO TEÓRICO

A continuación, se describirá el marco teórico que servirá como base para el estudio, es decir, conocimiento consolidado, conceptos técnicos, metodologías y herramientas que estarán involucradas en la solución propuesta.

2.1. Aprendizaje automático

El aprendizaje automático, más conocido como *Machine Learning* (ML), es un concepto fundamental dentro de la rama de la inteligencia artificial, "describe la capacidad de los sistemas para aprender de datos de entrenamiento específicos de problemas para automatizar el proceso de construcción de modelos analíticos y resolver tareas asociadas" [Janiesch *et al.*, 2021]. En esta subsección se hará un repaso general sobre los conceptos clave del ML.

2.1.1. Paradigmas de aprendizaje

Los paradigmas de aprendizaje en ML se refieren a los métodos generales que los sistemas utilizan para aprender a partir de los datos, y se dividen generalmente en tres dominios, aprendizaje supervisado, aprendizaje no supervisado y aprendizaje reforzado, los que abarcan una gran variedad de problemas. La figura 1, presenta la taxonomía de dos de los métodos como referencia.

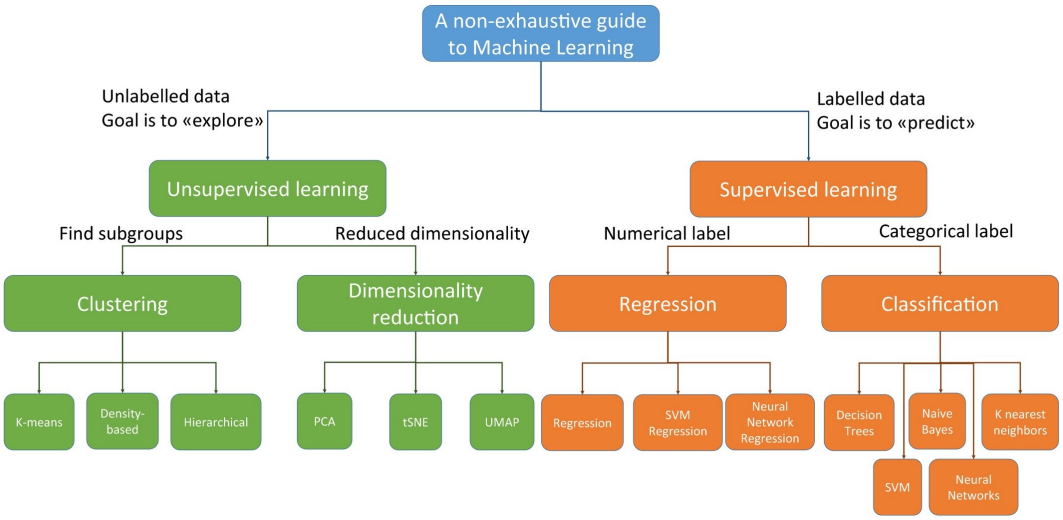


Figura 1: Taxonomía general de algoritmos de ML.
Fuente: [Badillo *et al.*, 2020].

Aprendizaje supervisado "En un problema de aprendizaje supervisado, la computadora recibe datos de entrenamiento con observaciones y los correspondientes valores de salida conocidos" [Badillo *et al.*, 2020]. Es el paradigma más común, donde el objetivo es inferir la salida para entradas no vistas mediante la identificación de patrones, tendencias o reglas generales aprendidas del conjunto de entrenamiento.

Aprendizaje no supervisado "En el análisis exploratorio de datos, a menudo no conocemos las verdaderas 'etiquetas' o es posible que queramos examinar los patrones que surgen naturalmente en los datos" [Badillo *et al.*, 2020]. Se trabaja con conjuntos de ejemplos para realizar tareas de agrupamiento, reducción de dimensionalidad o detección de anomalías. Al no saber las verdaderas 'respuestas correctas' no existen sesgos por parte del investigador, lo que permite aprender bastante de los datos que se están trabajando. Además, este tipo de aprendizaje elimina la necesidad de etiquetar los datos, lo cual puede ser una tarea que podría requerir mucho tiempo y que en muchos casos no es viable.

Aprendizaje por refuerzo Este paradigma está basado en un sistema de recompensa/castigo y puede ayudar a incrementar la automatización u optimizar la eficiencia en ambientes como la robótica, conducción autónoma o en la logística de cadenas de fabricación. "En términos generales, su principal objetivo es aprender a asignar estados a acciones mientras se maximiza una señal de recompensa" [Morales y Zaragoza, 2012].

2.1.2. XGBoost

eXtreme Gradient Boosting (XGBoost) es un sistema de *tree boosting*[Friedman, 2001] que es altamente efectivo y ampliamente utilizado en la comunidad de ciencia de datos, que "ha demostrado grandes resultados en varias aplicaciones de clasificación"[Chen y Guestrin, 2016]. XGBoost se caracteriza por su capacidad para manejar grandes volúmenes de datos y por incorporar diversas optimizaciones que mejoran tanto la velocidad como la precisión de los modelos generados, algunas de estas optimizaciones incluyen técnicas de regularización para evitar sobreajuste y uso de los beneficios que aportan la computación paralela y distribuida.

Tree boosting Es una técnica de ML de aprendizaje supervisado que combina múltiples modelos simples (*ensembles*), en este caso árboles de decisión, para construir un modelo fuerte y preciso. En XGBoost, los árboles se construyen de manera secuencial, donde cada nuevo árbol intenta corregir los errores de los árboles anteriores, lo que se conoce como aprendizaje aditivo.

2.1.3. Redes neuronales artificiales

Las *Artificial Neural Network* (ANN) son modelos computacionales inspirados en el cerebro humano que están diseñados para reconocer patrones y son capaces de modelar relaciones no-lineales. "Está formada por cientos de unidades individuales, neuronas artificiales (Figura 3) o elementos de procesamiento, conectadas con coeficientes (pesos), que constituyen la estructura neuronal y están organizadas en capas" [Agatonovic-Kustrin y Beresford, 2000]. Han sido exitosamente aplicadas en áreas como la visión por computadora, predicción de series de tiempo, entre otros.

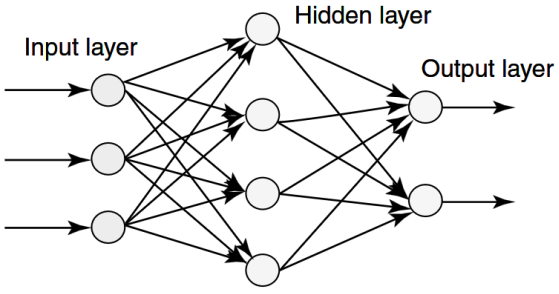


Figura 2: Red neuronal artificial multicapa.
Fuente: [Abraham, 2005].

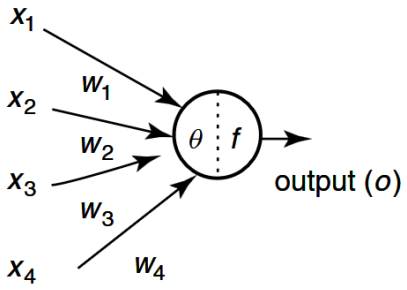


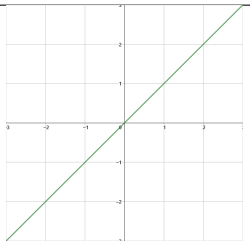
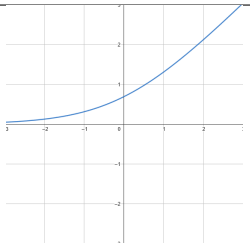
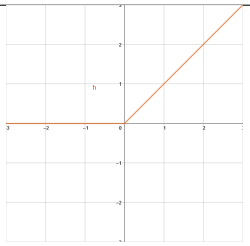
Figura 3: Neurona artificial.
Fuente: [Abraham, 2005].

Función de activación Una función de activación dentro de una ANN es una transformación matemática donde "calculamos la suma de los productos de las entradas y sus pesos corres-

pondientes y finalmente le aplicamos una función de activación para obtener la salida de esa capa en particular y suministrarla como entrada a la siguiente capa" [Sharma *et al.*, 2017]. Algunos ejemplos de función activación son las siguientes (Tabla 1):

Tabla 1: Funciones de activación

Fuente: Elaboración propia.

Nombre	**Gráfica**	$g(x)$	$g'(x)$	**Rango**
Identidad		x	1	$(-\infty, \infty)$
Softplus		$\log_e(1+e^x)$	$\frac{e^x}{1+e^x} = \frac{e^x}{1+e^{-x}}$	$[0, \infty)$
ReLU		$x^+ = max(0, x)$	$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$	$[0, \infty)$

Parámetro Los parámetros de una ANN son coeficientes como los pesos y sesgos que la red ajusta durante el entrenamiento. La optimización de los parámetros es fundamental para que la red aprenda adecuadamente de los datos de entrenamiento y realice predicciones precisas.

Retro-Propagación También conocido como *Backpropagation* es comúnmente usado para entrenar las ANN, este "calcula cómo cambiar ligeramente la intensidad de cada sinapsis, modificaría el error de la red, utilizando la regla de la cadena, ..., los cálculos de errores comienzan en la capa final y fluyen hacia atrás, lo que lleva a la noción de que los errores se 'propagan hacia atrás' a través de la red" [Lillicrap *et al.*, 2020]. La actualización de pesos W_{ij} para una capa oculta está dada por:

$$\Delta W_{ij} = -\eta \frac{\partial E}{\partial W_{ij}} = -\eta h_i \delta_j \text{ donde } \delta_j = \left(\sum_k \delta_k W_{jk} \right) f'(a_j) \tag{1}$$

Donde:

- η es la tasa de aprendizaje, controla cuánto se ajustan los pesos de la red en respuesta al error calculado.
- E es la función de error.
- h_i es la salida de la neurona i,$h_i = f(a_i)$, donde a_i es la entrada de la neurona.
- δ_k a veces llamado 'señales de error' y se computan recursivamente mediante la regla de la cadena. En la capa de salida $\delta_o = y_o - t_o$, donde y_o es la salida de la red y t_o es la salida objetivo.

2.1.4. Red Feed-forward

Una red *Feed-forward*[Bebis y Georgiopoulos, 1994] es una ANN donde la información fluye unidireccionalmente (Figura 2) sin ciclos (como ocurre en las redes neuronales recurrentes). Esta compuesta por tres tipos de capas:

- Entrada: Recibe los datos de entrada, donde cada neurona en esta capa representa una 'característica' de la entrada.
- Oculta: Esta ubicada entre la capa de entrada y salida. Puede tener varias o ninguna y su objetivo es aprender la representación interna de los datos.
- Salida: Es la capa final que produce el resultado de la inferencia.

Este tipo de red es especialmente adecuado para problemas donde la relación entre las características de entrada y la salida no depende de secuencias temporales o patrones recurrentes.

2.1.5. Monte Carlo Dropout

La red bayesiana *Monte Carlo Dropout*[Gal y Ghahramani, 2016] usa el método de regularización *Dropout*[Srivastava *et al.*, 2014] durante la fase de entrenamiento e inferencia, lo que permite estimar la incertidumbre[1] mediante el uso de distribuciones.

[1]La incertidumbre en este contexto se refiere a la variabilidad o falta de certeza en las predicciones realizadas por la red neuronal.

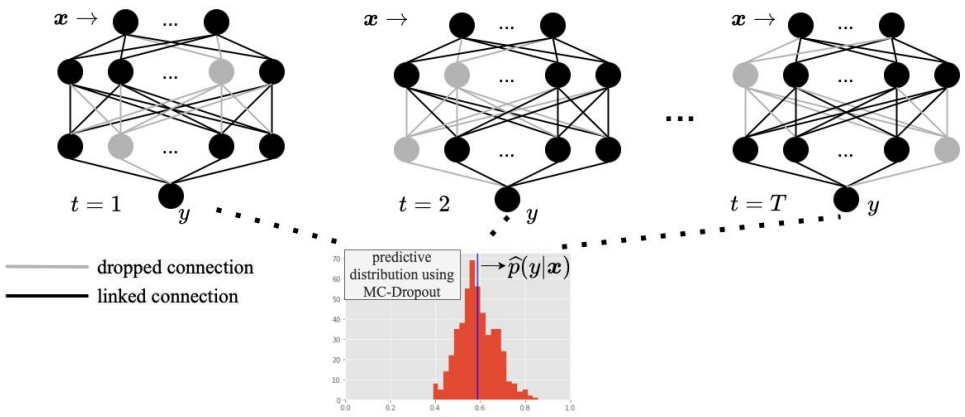


Figura 4: Red *Monte Carlo Dropout* usada para aproximar la distribución predictiva, obteniendo así la media predictiva.
Fuente: [Pezoa *et al.*, 2022].

Dropout "La idea clave es eliminar aleatoriamente unidades (junto con sus conexiones) de la red neuronal durante el entrenamiento."[Srivastava *et al.*, 2014] Esto previene en gran medida el *overfitting* y, al utilizarlo, se muestrean redes neuronales más 'delgadas' (Figura 6), además de proporcionar una forma de combinar de manera aproximada muchas arquitecturas diferentes mediante la eliminación de neuronas.

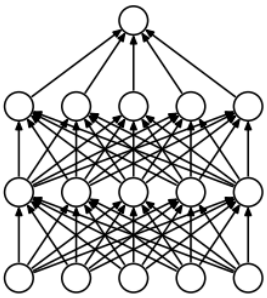


Figura 5: Red neuronal artificial estándar.
Fuente: [Srivastava *et al.*, 2014].

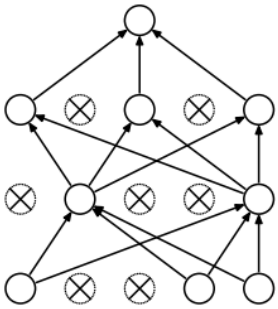


Figura 6: Luego de aplicar *dropout*. Las neuronas con cruces han sido eliminadas.
Fuente: [Srivastava *et al.*, 2014].

Al usar *Monte Carlo Dropout* durante la fase de inferencia, en lugar de desactivar el *dropout* como es común en redes estándar, se mantiene activo y se realizan múltiples predicciones de la misma entrada, cada vez con un conjunto diferente de neuronas desactivadas, simulando una colección de modelos distintos. "Para estimar la media y la incertidumbre predictiva, simplemente recopilamos los resultados de los *stochastic forward passes* a través del modelo"[Gal y Ghahramani, 2016]. La diferencia que presenta esta técnica respecto a otro tipo de redes bayesianas, como las basadas en infe-

rencia variacional [Hoffman *et al.*, 2013, Titsias y Lázaro-Gredilla, 2014, Paisley *et al.*, 2012, Rezende *et al.*, 2014, Kingma y Welling, 2013] es que "para representar la incertidumbre, el número de parámetros en estos modelos se duplica para el mismo tamaño de red" [Gal y Ghahramani, 2016].

2.1.6. TabNet

TabNet es una arquitectura de red neuronal diseñada específicamente para manejar datos tabulares, un tipo de datos que para redes neuronales "sigue siendo poco explorado, con variantes de árboles de decisión en ensamblaje que aún dominan la mayoría de las aplicaciones"[Arik y Pfister, 2021]. Esta arquitectura introduce un enfoque que combina atención secuencial, lo que permite al modelo enfocarse dinámicamente en diferentes subconjuntos de características en distintas etapas de la predicción, y el enmascaramiento de características permitiendo que solo ciertas características sean consideradas en cada paso de la inferencia. Este enfoque no solo mejora la precisión del modelo al reducir el ruido, sino que también contribuye significativamente a la interpretabilidad del modelo.

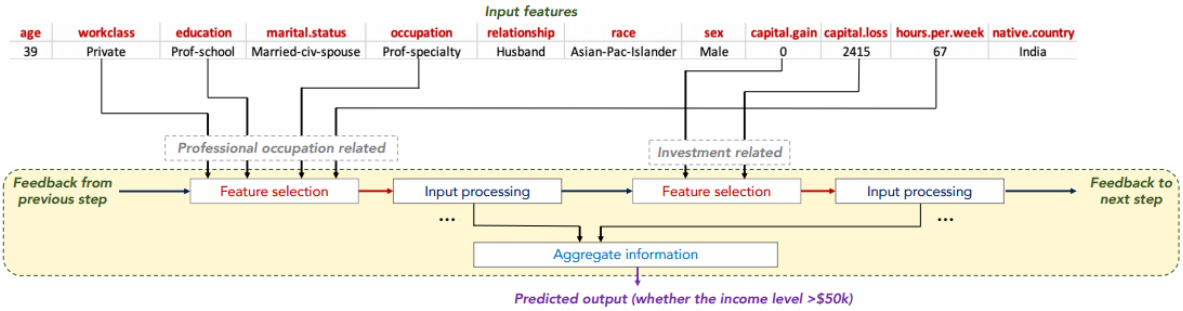


Figura 7: Ejemplo de selección de características dispersas para la predicción de ingresos. Se emplean múltiples bloques que se centran en procesar un subconjunto de características de entrada para el razonamiento. "Dos bloques de decisión mostrados como ejemplos procesan características relacionadas con la ocupación profesional y las inversiones, respectivamente, para predecir el nivel de ingresos".

Fuente: [Arik y Pfister, 2021].

2.2. SHAP

Entender como funcionan los modelos de ML es esencial para respaldar la toma de decisiones de los usuarios con respecto a los resultados entregados por el modelo.

Las explicaciones como *SHapley Additive exPlanations* (SHAP)[Lundberg y Lee, 2017] nos da claridad sobre como contribuye cada variable a la salida del modelo, esto lo consigue mediante el uso de los *Shapley values* [Shapley, 1951] de la teoría de juegos.

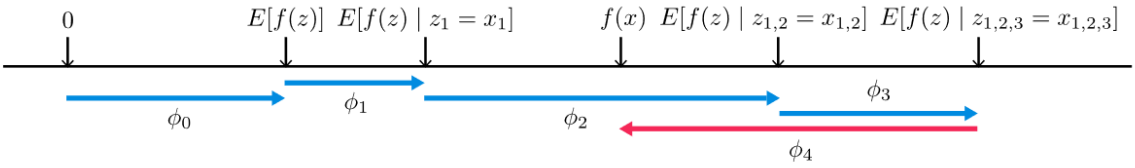


Figura 8: Ejemplo de importancias de variables en SHAP.
Fuente: [Arik y Pfister, 2021].

En la Figura 8, se explica como "se llega desde el valor base $E[f(z)]$ que se predeciría si no conociéramos ninguna característica hasta la salida actual $f(x)$"[Lundberg y Lee, 2017].

SHAP puede ofrecer explicaciones tanto a nivel global (cómo las características afectan las predicciones del modelo), como a nivel local (cómo una característica específica afecta la predicción de un caso particular).

2.3. Virtualización

La virtualización es una tecnología que permite crear versiones virtuales de recursos físicos, como servidores, dispositivos de almacenamiento, redes y hasta sistemas operativos completos. "Combina o divide recursos informáticos para presentar uno o varios entornos operativos utilizando metodologías como partición o agregación de *hardware* y *software*, simulación parcial o completa de máquinas, emulación, tiempo compartido y otras" [Chiueh y Brook, 2005].

Una técnica de virtualización es la virtualización a nivel de *Operating System* (OS), donde el kernel del OS permite la existencia de múltiples instancias aisladas y seguras, conocidas como contenedores. Estas instancias, aunque funcionan como sistemas operativos independientes, comparten el mismo kernel anfitrión.

Docker Es una plataforma que ayuda a los desarrolladores a construir, compartir y ejecutar aplicaciones dentro de contenedores que están empaquetadas dentro de 'imágenes' que incluyen todas sus dependencias. "La *containerization* permite al usuario ejecutar aplicaciones en un entorno virtual empaquetando todos los elementos necesarios, como archivos, bibliotecas y otros componentes esenciales" [Docker, 2020], algunas de las ventajas del uso de esta tecnología son su rapidez y flexibilidad, ya que los contenedores `Docker` contienen solo los datos que son necesarios para la aplicación, esto los hace bastante compactos en comparación con las máquinas virtuales (VM), además ofrecen portabilidad para el *deploy* de aplicaciones, ya que se pueden ejecutar en diferentes máquinas y ambientes sin una configuración adicional extensiva.

Apptainer Al igual que *Docker*, es una plataforma que permite crear y ejecutar contenedores. La principal diferencia radica en su enfoque y uso en *clusters* de computación de alto desempeño (HPC) y en la seguridad, esto se debe a que no requiere privilegios de superusuario para ejecutar contenedores, lo que es esencial en entornos HPC donde el acceso de superusuario está restringido, además los contenedores *Apptainer* se empaquetan en un solo archivo de formato `SIF`, lo que facilita su transporte y ejecución en diferentes sistemas, al contrario de *Docker* que utiliza una estructura de capas que puede requerir más configuración para garantizar la compatibilidad entre diferentes entornos.

2.4. Herramientas de medición de recursos computacionales

Antes de abordar las herramientas de medición, se definen por recursos computacionales las métricas como: el tiempo de ejecución, memoria RAM utilizada, IOPS, energía consumida, entre otros.

Otro elemento fundamental para la medición de recursos computacionales son las llamadas al sistema, pues actúan como un puente entre el *software* y el *hardware*, permitiendo medir cómo el *software* utiliza los recursos físicos del *hardware*. “Algunas llamadas al sistema tienen una relación obvia con ciertos tipos de recursos. Por ejemplo, las llamadas al sistema de escritura, lectura y similares funcionan en archivos o *sockets*, lo que se traduce en E/S de disco o de red”[Schmidt *et al.*, 2018].

strace Es una herramienta disponible en sistemas UNIX que permite rastrear llamadas al sistema. Su carga en el sistema es baja y se utiliza para monitorear y alterar las interacciones entre procesos y el kernel de Linux, que incluyen llamadas al sistema y cambios de estado de procesos.

ltrace Cuando se requiere un mayor nivel de detalle `ltrace` puede ser usado para interceptar y registrar llamadas dinámicas a librerías. También puede interceptar e imprimir las llamadas al sistema ejecutadas por el programa.

perf Es una herramienta de análisis de rendimiento y depuración de Linux. Permite la recopilación de datos sobre una amplia variedad de eventos, tanto del *hardware* (como ciclos CPU, caché, acceso a memoria) como del *software* (llamadas al sistema).

CAPÍTULO 3
ESTADO DEL ARTE

El trabajo actual usa como base el artículo "*Representation Learning for Resource Usage Prediction*" [Schmidt *et al.*, 2018] donde se diseñó una red *Long Short-Term Memory* (LSTM)[2] (en figura 9), entrenada específicamente para predecir el uso de recursos futuro de aplicaciones en ejecución. La red utiliza los estadísticos de rendimiento, como la utilización de la CPU, la cantidad de *bytes* escritos en el disco, en conjunto con la representación vectorial de las llamadas al sistema hechas por la aplicación, usando una conversión similar a *word2vec Skip-gram model*[Mikolov *et al.*, 2013]. El objetivo es crear un modelo que aproxime el estado de un sistema en cualquier momento y permitir resolver tareas como predicción del uso de recursos y detección de anomalías.

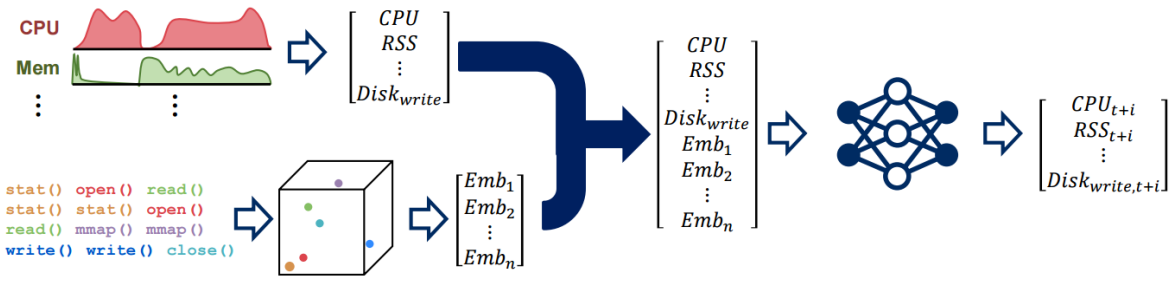


Figura 9: Red LSTM para análisis de recursos computacionales.
Fuente: [Schmidt *et al.*, 2018].

Para predecir el uso de recursos (figura 9) se recolectan datos de telemetría típicos (figura 9: arriba a la izquierda), como llamadas al sistema de la aplicación (figura 9: abajo a la izquierda), durante un período de tiempo t. El número variable de llamadas se transforma en un vector de tamaño t fijo mediante *word embedding*. Los dos vectores se combinan y se utilizan como entrada para una red LSTM que predice el uso de recursos en algún período de tiempo futuro $t + i$.

Uno de los estudios que se usarán como comparación del trabajo actual será "*On the Fair Comparison of Optimization Algorithms in Different Machines*"[Arza *et al.*, 2023] que estudia la comparación experimental del rendimiento de algoritmos de optimización en dos máquinas distintas, entre un nuevo algoritmo `B` contra un algoritmo conocido `A` si solo tenemos los resultados y tiempos de ejecución en cada instancia del algoritmo `A`. Donde se plantea una metodología que consta de dos partes para realizar la comparación:

1. Un modelo de estimación de dos parámetros que, "dado el tiempo de ejecución de un algoritmo en una máquina, estima el tiempo de ejecución del mismo algoritmo

[2]Es una red neuronal recurrente con memoria de estado y estructura celular multicapa.

en otra máquina. Este modelo puede ajustarse de modo que la probabilidad de estimar un tiempo de ejecución más largo de lo que debería sea arbitrariamente baja" [Arza *et al.*, 2023].

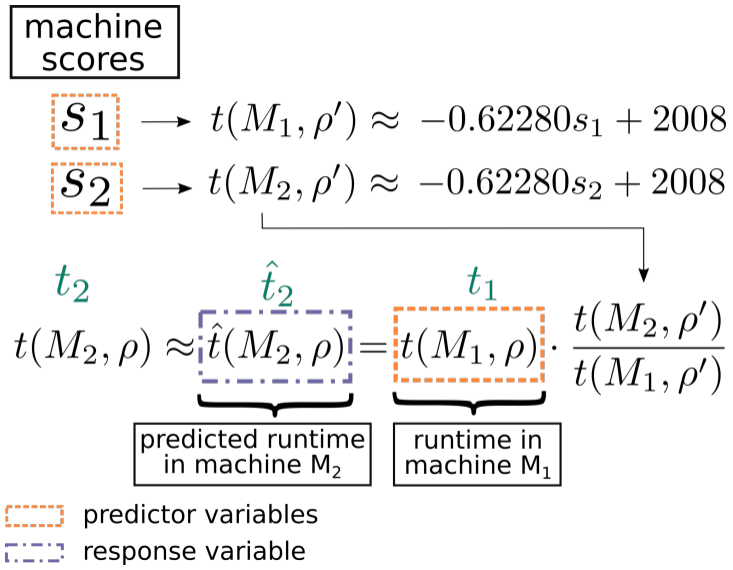


Figura 10: La estimación es llevada a cabo por tres variables, las puntuaciones *single-thread* de los procesadores en *Passmark* s_1 y s_2 y el tiempo total de ejecución del problema de optimización en la máquina base $t(M_1, \rho)$.
Fuente: [Arza *et al.*, 2023].

2. Usando una adaptación de *one-sided sign test* [Conover, 1999] que utiliza un *p-value* modificado y tiene en cuenta la probabilidad del punto anterior. Dicha adaptación evita aumentar la probabilidad de error de falsos positivos asociada con la ejecución de los algoritmos `A` y `B` en diferentes máquinas.

El trabajo actual se comparará con la primera parte de la metodología.

Los siguientes estudios están ordenados cronológicamente y contribuyen a la contextualización del estado actual de la problemática.

Anteriormente, ya se habían estudiado diversas técnicas de aprendizaje automático para la predicción de consumo de recursos computacionales, como en el artículo "*On the use of machine learning to predict the time and resources consumed by applications*"[Matsunaga y Fortes, 2010] en el que se realiza la comparación entre modelos como *K-Nearest Neighbor* (KNN), Regresión lineal, *Support Vector Machine* (SVM) [Drucker *et al.*, 1996], árboles de decisión, ANN, *Predicting Query Runtime* (PQR) [Gupta *et al.*, 2008] y un nuevo método basado en PQR llamado *Predicting Query Runtime Regression* (PQR2), donde se plantea que predecir el consumo de CPU, RAM, uso de disco y tiempo de ejecución puede ser visto como un problema de ML de aprendizaje supervisado, ya que "el sistema necesita aprender un concepto basado en una colección de datos

históricos de n ejecuciones anteriores de la aplicación" [Matsunaga y Fortes, 2010]. El nuevo método (PQR2) es una generalización de PQR, que añade funciones de regresión a las hojas del árbol de decisión generado por PQR para obtener una predicción más refinada. También se investigó el impacto de los atributos (consumo de CPU, RAM, entre otras) en la precisión de la predicción. Para el ambiente de pruebas se usaron cuatro computadores distintos, donde se usaron dos aplicaciones populares del área de bioinformática llamadas *Basic Local Alignment Search Tool* y *Randomized Axelerated Maximun Likelihood*. Tiempo de ejecución, cantidad de memoria RAM ocupada y el tamaño de la salida del programa fueron las variables objetivo de la predicción, donde como entrada se tomaron en cuenta las variables especificas de cada aplicación, el nombre del *cluster* y los recursos del sistema (reloj de CPU, cantidad de cache y cantidad de RAM). Aunque según el artículo, información como el reloj de CPU no refleja bien la capacidad de procesamiento de cada recurso, por lo que se usaron los siguientes *benchmarks* para visualizar de mejor forma cada recurso:

- Una aplicación de multiplicación de matrices fue usada para extraer la cantidad de operaciones de punto flotante realizadas en una cantidad fija de tiempo para conocer la **velocidad del CPU**.
- Una aplicación de *benchmark* de cache fue usada para obtener el rendimiento de las operaciones de lectura, modificación y escritura, estas operaciones componen la **velocidad de la memoria**.
- El comando de linux `dd` fue usado para leer un archivo de 1 GiB de diferentes fuentes, lo que permitió conocer la **velocidad de lectura de disco**.

El procesamiento de estos atributos es clave para una mejor representación de las características de los equipos y será un tema a tratar en próximas secciones. Los resultados experimentales de la comparación demostraron que en cuanto a precisión PQR2 fue el que se pudo adaptar mejor a los diferentes escenarios, aun así tanto SVM como KNN presentaron resultados competitivos, siendo el principal problema de KNN encontrar un número ideal de vecinos, "especialmente cuando la información de entrenamiento presenta una densidad variable de puntos a través del espacio de datos"[Matsunaga y Fortes, 2010].

Dentro del contexto de computación en la **nube** la predicción de consumo de recursos computacionales es un tópico atractivo, tal como lo demuestra el estudio "*A Resource Usage Prediction System Using Functional-link and Genetic Algorithm Neural Network for Multivariate Cloud Metrics*"[Nguyen *et al.*, 2018] donde se presenta un sistema *cloud* de predicción usando *Functional Link Neural Network* (FLNN)[Pao, 1989] entrenada con la ayuda de un algoritmo genético en orden de incrementar la eficiencia de la predicción. Se comenta que el procesamiento de múltiples tipos de métricas (uso de CPU, uso de memoria, entre otras) al mismo tiempo ha recibido una menor atención por los investigadores al desarrollar sistemas de predicción, lo que ignora el hecho de que pueden existir relaciones implícitas entre las distintas métricas al momento de realizar predicciones. De esta manera, los resultados de la inferencia podrían no ajustarse a la realidad. Este estudio es innovador porque presenta un

avance en cuanto a realizar predicciones tomando en cuenta múltiples métricas simultáneamente, buscando tomar en cuenta estas relaciones. La red FLNN fue elegida, ya que contiene solo una neurona, tiene un menor costo computacional y es fácil de implementar en comparación con las *Multi-layer neural networks* presentes en *deep learning*. La razón de esto es porque "la capa oculta en esta red se elimina. De esta manera, la relación no lineal entre las entradas y las salidas se procesa mediante un conjunto de polinomios."[Nguyen *et al.*, 2018]. El algoritmo genético apoya a FLNN resolviendo las desventajas del *back propagation* tradicional. Para probar este modelo se generó un *dataset* con la ayuda de *Google cluster*.

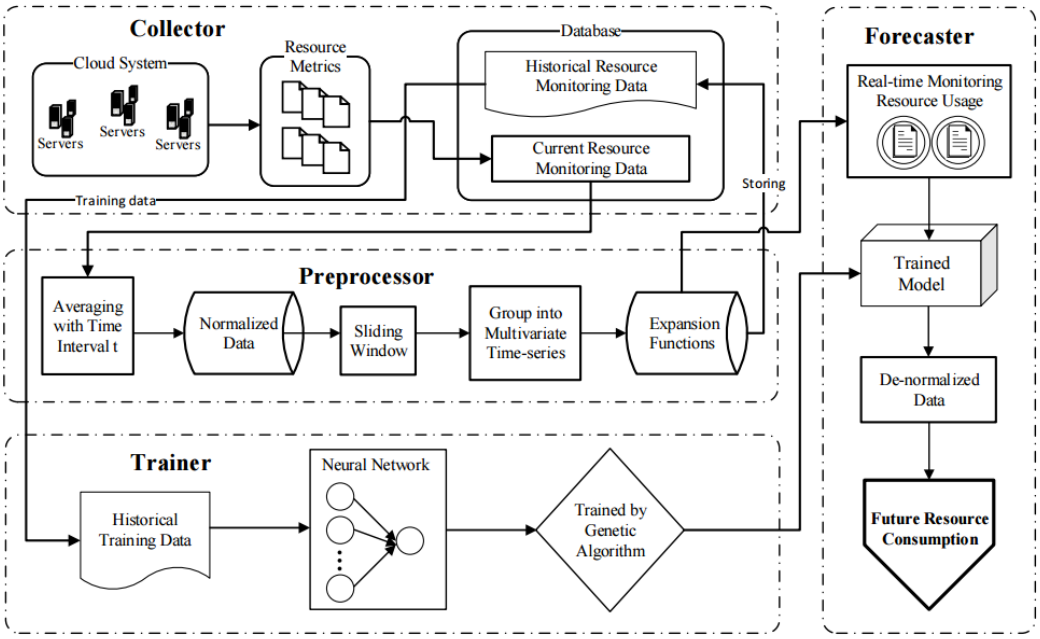


Figura 11: Sistema de predicción de recursos multivariable.
Fuente: [Nguyen *et al.*, 2018].

El diseño del sistema de predicción del estudio (figura 11) se compone de cuatro módulos principales. El módulo *Collector* que extrae las métricas desde las VM y las guarda en un repositorio. El módulo *Preprocessor* transforma los datos en bruto para que se ajusten a la entrada de la red neuronal.

El módulo *Trainer* contiene el modelo FLNN entrenado por el algoritmo genético para acelerar la convergencia e incrementar la precisión, por la combinación de estas técnicas el método de aprendizaje propuesto es llamado FL-GANN, este módulo se divide en dos componentes principales. La figura 12 "describe el proceso de entrenamiento, en donde el componente *encoder* es usado para codificar los pesos y sesgos de la red en un cromosoma (vector). Mientras que el componente *decoder* decodifica el cromosoma en pesos y sesgos de la red"[Nguyen *et al.*, 2018]. Donde luego el modelo entrenado sea consumido en el módulo *Forecaster*. Con este sistema se consigue un buen rendimiento en el proceso de predicción, mientras que se aprovechan las relaciones implícitas entre las distintas métricas.

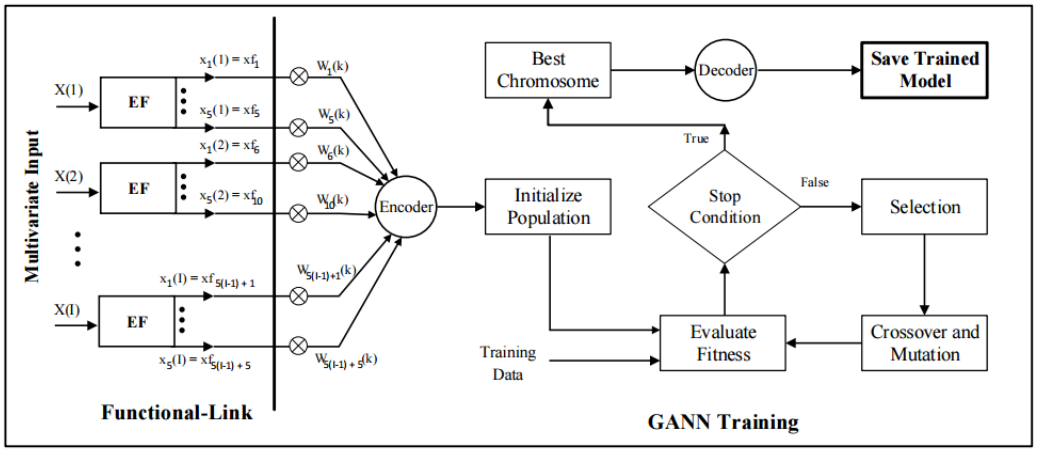


Figura 12: Proceso de entrenamiento FL-GANN.
Fuente: [Nguyen *et al.*, 2018].

La computación en la nube se ha vuelto una parte integral de las infraestructuras tecnológicas y una de sus problemáticas es la predicción de recursos, se ha logrado un gran avance en la investigación a partir de este contexto en específico, al igual que el anterior estudio, el siguiente artículo "*Deep Learning-Based Prediction of CPU and Memory Consumption for Cost-Efficient Cloud Resource Allocation*"[Dittakavi, 2021] ofrece un sistema de predicción de consumo de recursos computacionales en la nube, desde la perspectiva del *deep learning*. El objetivo del modelo es facilitar la asignación de recursos mediante la predicción de uso de CPU y memoria RAM mediante el uso de *Temporal Fusion Transformer* (TFT) y *Gated Recurrent Units* (GRU). Fue entrenado en el *dataset Google Cluster Workload Traces 2019*. TFT y GRU fueron comparados contra un modelo más tradicional llamado *Exponential Smoothing* (ETS). El estudio parte de la primicia que en la nube "la CPU y la memoria son dos de los principales recursos que determinan el rendimiento y la eficiencia de las aplicaciones y servicios"[Dittakavi, 2021] y que una asignación eficiente de estos dos recursos es vital porque permiten que una aplicación se ejecute adecuadamente, sin degradar la experiencia de usuario. Por otra parte, asignar holgadamente tanto CPU como memoria pueden derivar en un incremento de costos sin algún beneficio en el rendimiento. El estudio está enfocado en el ambiente de la nube, donde los proveedores proveen típicamente herramientas de perfilado con varias métricas, en este caso las métricas usadas serán los usos de CPU y memoria RAM, además de otros datos relacionados con la red, para el procesamiento de datos se interpolaron los valores perdidos y se trataron los *outliers*. La evaluación de resultados experimentales demostró que tanto el modelo GRU como el modelo TFT superaron al enfoque tradicional con el modelo ETS, siendo el TFT él con mejor rendimiento entre los tres con una *Mean Absolute Error* (MAE) con valor $5{,}23$ comparado con los $\approx 10{,}67$ para ETS y $6{,}45$ para GRU. De este artículo se puede rescatar principalmente el uso de técnicas cada vez más modernas de ML y, en conjunto con el anterior estudio analizado, la importancia que tiene la mejora de la predicción del uso de recursos en el contexto de la nube.

La principal métrica a abarcar en la investigación de un modelo que pueda predecir el con-

sumo de recursos computacionales es el tiempo de ejecución del programa, que indicará cuando se liberaran los recursos actualmente usados. Predecir solo esta métrica puede ser visto como un subproblema de alto interés, el estudio "*Estimation of execution time for computing tasks*"[Bielecki y Śmiałek, 2023] se centra en la estimación del *average-case execution time* (ACET) mediante modelos de ML tradicionales como SVM, KNN o regresión polinomial, usando *data* histórica, dentro de un ambiente de contenedores llamados *Computation Modules*. El enfoque que se decidió tomar fue tratar al programa como un *black box*, es decir, no se permiten extracción de características desde el código, por lo que se asumirá que el código de la aplicación no estará disponible. El modelo recibirá propiedades tanto de la entrada del programa como del ambiente de ejecución históricas. Como la estimación de tiempo es un problema bastante estudiado se han propuesto varios enfoques, el usado en el estudio es el enfoque estadístico, donde el tiempo de ejecución es tratado como una variable aleatoria y es estimado a base de observaciones pasadas. Los datos de entrenamiento fueron generados durante varias ejecuciones de los contenedores. Cada observación se compuso por el siguiente conjunto de variables:

- CPUS: La fracción de un CPU físico usado para llevar a cabo la ejecución de un contenedor.
- OVER: El tamaño total de la entrada en *bytes*.
- PART: El número de elementos en la entrada.
- AVG: El promedio de tamaño de la entrada.
- MAX: El máximo de tamaño de la entrada.

Al analizar los resultados del estudio se concluye que la regresión polinomial ($65{,}3\,\%$) fue la con peor desempeño, teniendo KNN ($43{,}7\,\%$) y SVM ($40{,}8\,\%$) un rendimiento similar en cuanto al error relativo. Siendo SVM el más prometedor, ya qué permite estimar la ejecución de tiempo de una forma más general.

En la actual investigación se plantea el uso de las métricas recolectadas después de la ejecución del programa en una estructura tabular, por lo que es importante tener en cuenta las técnicas que actualmente ofrecen un buen desempeño para estos tipos de datos. Esto es lo que se aborda en el estudio "Tabular Data: Deep Learning Is Not All You Need"[Shwartz-Ziv y Armon, 2022], donde se investiga la efectividad de los modelos DP en comparación con los modelos basados en árboles de decisión, como XGBoost, que son comunes en muchas aplicaciones del mundo real. El artículo realiza una comparación entre modelos DP recientemente propuestos como TabNet[Arik y Pfister, 2021], NODE[Popov *et al.*, 2019] y DNF-Net[Katzir *et al.*, 2020], contra XGBoost[Chen y Guestrin, 2016]. A través de esta comparación, se observa que XGBoost supera a los modelos DP en la mayoría de los casos. Además, se demuestra que un *ensamble* que combina XGBoost con modelos DP proporciona los mejores resultados, superando tanto a XGBoost como al resto de modelos. En el artículo se concluye que aunque los modelos

DP han mostrado avances significativos, no reemplazan a los modelos basados en árboles de decisión como XGBoost. Además, se destaca la importancia de considerar enfoques híbridos que combinen a los modelos tradicionales con DP para maximizar el rendimiento de inferencia en datos tabulares.

CAPÍTULO 4
PROPUESTA DE SOLUCIÓN

En la siguiente propuesta de solución se estableció el tiempo de ejecución en ejecuciones del tipo *single-thread* como principal métrica a inferir a partir de las distintas estadísticas de consumo y especificaciones de *hardware* que se mencionarán en las próximas secciones.

Como se mencionó en el estado del arte, este estudio se basó en el artículo "*Representation Learning for Resource Usage Prediction*" [Schmidt *et al.*, 2018], principalmente en el flujo de recolección de datos, la mayoría de los estadísticos utilizados y el uso de llamadas al sistema para realizar la predicción. Sin embargo, a diferencia del enfoque en tiempo real presentado en |el artículo, en este estudio no se utilizaron las métricas para inferir el estado del sistema en un momento futuro, en lugar de ello, se recopilaron las métricas de una ejecución completa para predecir el tiempo de ejecución del programa en un sistema objetivo, que es diferente del sistema base, en donde se obtuvieron las métricas. Además, las métricas se almacenaron en un formato tabular y se utilizó un conteo de las llamadas al sistema en lugar de *embeddings*.

4.1. Obtención de datos

Para la extracción de datos inicial se generó un conjunto de 92 pruebas, donde se aborda la resolución de distintas instancias de los siguientes problemas de optimización:

Traveling Salesman Problem Estudiado tanto en investigación de operaciones y ciencia de computación teórica, el problema del vendedor viajero (TSP), se plantea como: "dada una lista de ciudades y sus distancias por pares, la tarea es encontrar el recorrido más corto posible que visite cada ciudad exactamente una vez"[Goyal, 2010]. Para las pruebas se usaron las instancias del tipo `EUC_2D` de la biblioteca TSPLIB95[Reinelt, 1995].

Knapsack Problem La versión clásica del problema de la mochila se define como: "Se nos da un conjunto de n elementos, donde cada elemento j tiene un beneficio entero p_j y un peso entero w_j. El problema consiste en elegir un subconjunto de los elementos de manera que se maximice el beneficio total, mientras que el peso total no exceda una capacidad dada c"[Pisinger, 2005]. Se utilizaron varias de las instancias descritas en el estudio "*Where are the hard knapsack problems*"[Pisinger, 2005], tanto de coeficientes pequeños como algunas instancias difíciles.

N-Queens El problema de las N-Reinas es una variante del problema de las 8 reinas donde se plantea "cuántas formas hay de colocar n reinas en un tablero de ajedrez de $n \times n$ de manera que ninguna de las reinas se ataque entre sí"[Rivin *et al.*, 1994]. Para las pruebas se utilizó $n \in \{13; 14; 15\}$.

Además, se realizó la medición de métricas al realizar una multiplicación de matrices aleatorias de dimensión $\mathbb{R}^{n \times n}$ con $n \in \{1000; 1250; 1500; 1750; 2000; 2250; 2500\}$.

Todos los *solvers* se programaron en `C++` con la ayuda de `OR-Tools`, que es una *suite* de código abierto dedicada a la optimización combinatoria.

Se obtuvieron un total de 31 métricas, las cuales se dividieron en tres categorías: métricas de ejecución, especificaciones del *hardware* y conteo de llamadas al sistema (la explicación de cada métrica se encuentra en el Anexo 7.1). El proceso de recopilación de estas métricas de cada prueba se realizó con un programa escrito en `Python`, encargado de la ejecución de las pruebas como subprocesos, lo que facilita la obtención de las métricas de manera eficiente.

En el programa principal, se introduce la prueba a medir, donde se corre múltiples veces para la obtención de las métricas de ejecución relacionadas con el tiempo total de ejecución y el uso de la CPU, además se realiza una ejecución adicional para un perfilado detallado del uso de memoria. Para recopilar las especificaciones del *hardware* se ocupó el paquete `py-cpuinfo` y se obtuvieron manualmente las puntuaciones *Passmark* de cada computador utilizado desde el repositorio del estudio "*On the Fair Comparison of Optimization Algorithms in Different Machines*"[Arza *et al.*, 2023]. Finalmente, para el conteo de llamadas al sistema, se obtuvieron las llamadas de sistema del programa mediante el uso del comando `strace`, y posteriormente se contabilizó cuántas veces se realizaba cada llamada en la prueba.

Para garantizar la creación de un conjunto de datos diverso y representativo, se utilizaron dos conjuntos de computadores para realizar mediciones: el *cluster* de computación de alto rendimiento *Savio*, facilitado por la Universidad de California, Berkeley, y conjunto de computadores personales, donde se utilizaron 8 computadores del conjunto de *cluster Savio* y 6 del conjunto de computadores personales.

Para llevar a cabo cada prueba en un ambiente homogéneo y controlado, se hizo uso de herramientas de virtualización, en específico se utilizó *Apptainer* para el *cluster* de computación de alto rendimiento *Savio* y *Docker* para el conjunto de computadores personales. Las especificaciones de los computadores del *cluster Savio* y computadores personales utilizados se pueden ver en la Tabla 3 y 4 respectivamente.

4.2. Creación del conjunto de datos

Con la fase de extracción de datos ya completa, se realizaron las siguientes operaciones para crear el conjunto de datos con el que se entrenaran los modelos.

1. Se estandarizaron los tamaños de las memorias cache y el uso de RAM máximo de kilobytes a megabytes, para las frecuencias de los procesadores se transformó de hercios a gigahercios.

2. Se unieron las métricas de ejecución, especificaciones de *hardware* y conteo de llamadas al sistema en un solo conjunto mediante el criterio de ser el mismo procesador[3] y la misma prueba[4] ejecutada.

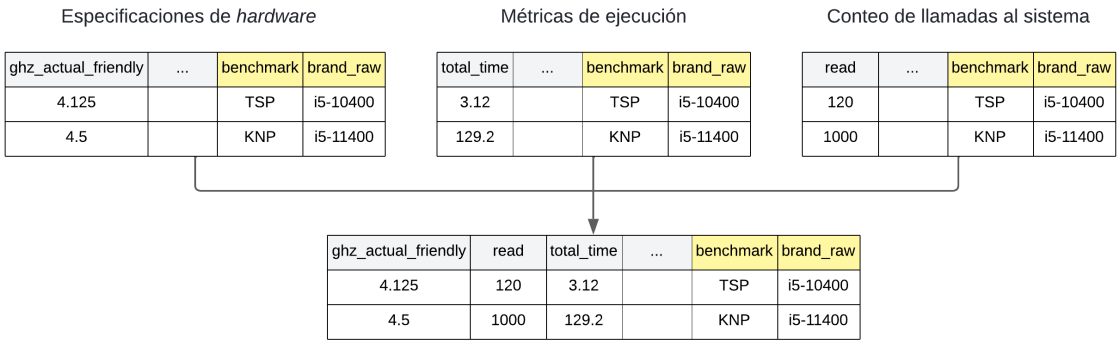


Figura 13: Creación de conjunto de datos, unión de variables.
Fuente: Elaboración propia, en *Lucidchart*.

3. Para obtener el conjunto de datos final se realizó una operación *self-join* en el conjunto de datos con el criterio de que sea la misma prueba, pero distinto computador, así obteniendo el tiempo final de la ejecución y características del computador objetivo (la explicación de las nuevas métricas se encuentra en el Anexo 7.2).

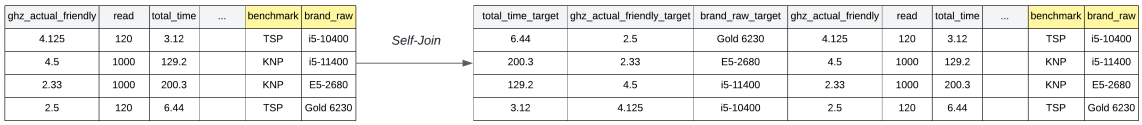


Figura 14: Creación de conjunto de datos, *self-join*.
Fuente: Elaboración propia, en *Lucidchart*.

[3] *brand_raw*: procesador
[4] *benchmark*: prueba

4.3. Análisis exploratorio de datos

Una vez construido el conjunto de datos, se le realizó un análisis exploratorio (EDA), donde se observó una gran concentración de pruebas con un tiempo de ejecución muy pequeño (Figura 15), lo que sugiere que usar el tiempo en escala logarítmica podría ser beneficioso al momento de representar los datos y entrenar los modelos.

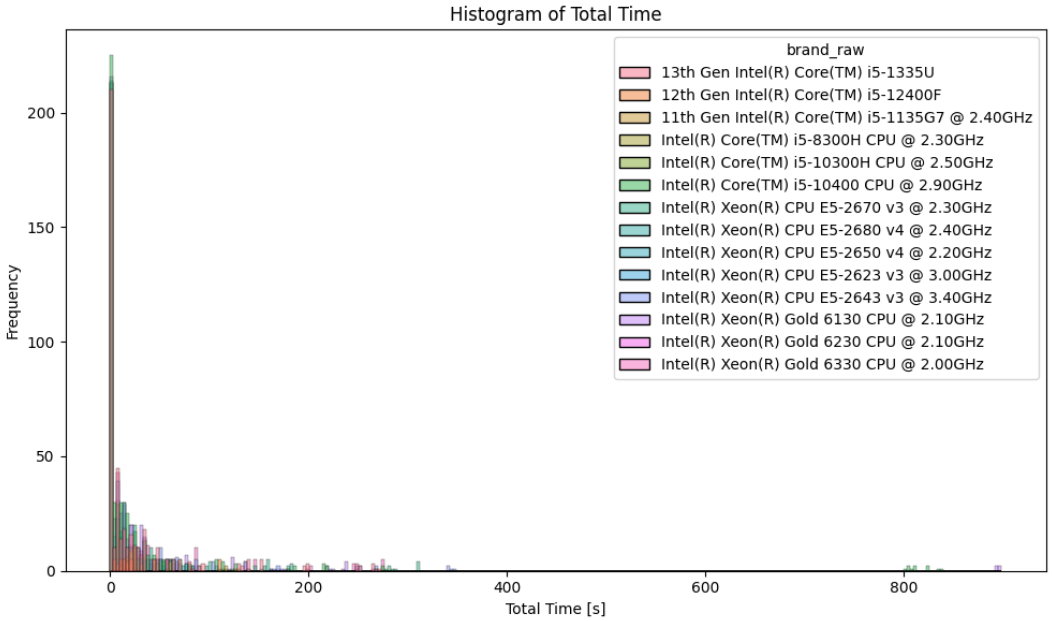


Figura 15: Histograma de tiempo de ejecución.
Fuente: Elaboración propia.

Luego se buscaban posibles relaciones lineales mediante el uso del coeficiente de correlación de Pearson, que "mide la dependencia lineal entre dos variables"[Sedgwick, 2012], donde el rango del coeficiente r varía en el intervalo $[-1; 1]$, indicando que:

- Si $r > 0$, existe una correlación lineal positiva, lo que implica que, a medida que una variable aumenta, la otra tiende a aumentar de manera proporcional.
- Si $r < 0$, existe una correlación lineal negativa, lo que implica que, a medida que una variable aumenta, la otra tiende a disminuir de manera inversamente proporcional.
- En general se buscan correlaciones fuertes ($r > |0{,}7|$), que indican una relación lineal más marcada.

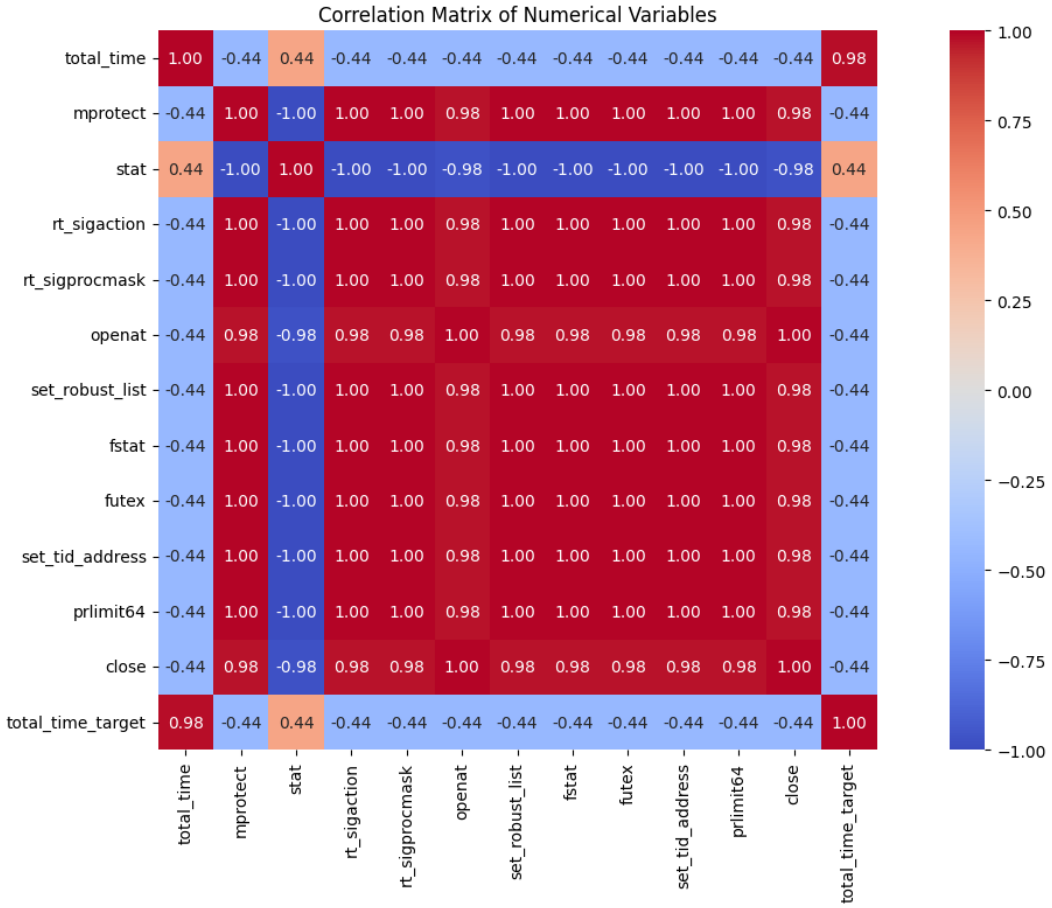


Figura 16: Matriz de correlación de Pearson. Solo se muestran las columnas con un coeficiente de correlación mayor a $0{,}4$ contra la variable objetivo.
Fuente: Elaboración propia.

Al analizar la matriz de correlaciones (Figura 16) para la variable objetivo *total_time_target*, que indica el tiempo total de ejecución del programa en la máquina objetivo, se determinó que solo tiene una relación lineal positiva muy fuerte con la variable *total_time*, que indica el tiempo total de ejecución del programa en la máquina base. En cambio, para las demás variables no existe una relación lineal fuerte, pero aún podrían existir relaciones no lineales entre ellas. Además, se observa que todos los conteos de las llamadas al sistema tienen una alta correlación entre sí, lo que puede indicar que estas instrucciones se utilizan juntas regularmente durante la ejecución del programa y que el uso de una llamada al sistema implica el uso de las demás.

Para examinar las correlaciones no lineales entre variables, se usarán los coeficientes de Spearman y Kendall, que en conjunto permiten identificar ciertos patrones de dependencia no lineal que no se manifiestan en la correlación lineal de Pearson.

El coeficiente de correlación de Spearman “evalúa qué tan bien una función monótona arbi-

traria puede describir la relación entre dos variables, sin hacer ninguna suposición sobre la distribución de frecuencia de las variables"[Hauke y Kossowski, 2011]. Al igual que el coeficiente de correlación de Pearson, ρ varía en $[-1; 1]$ indicando que:

- Si $\rho > 0$ existe una correlación monótona positiva, lo que implica que, a medida que una variable aumenta, la otra también tiende a aumentar de manera consistente, aunque no necesariamente de manera lineal.
- Si $\rho < 0$ existe una correlación monótona negativa, lo que implica que, a medida que una variable aumenta, la otra tiende a disminuir de manera consistente, aunque no necesariamente de manera lineal.

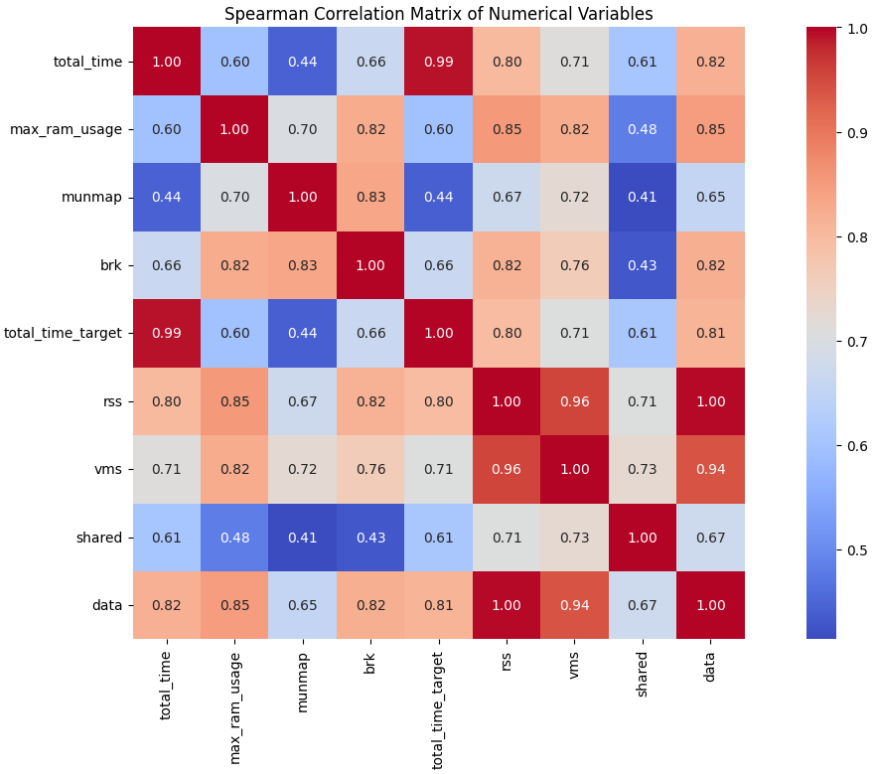


Figura 17: Matriz de correlación de Spearman. Solo se muestran las columnas con un coeficiente de correlación mayor a $0{,}4$ contra la variable objetivo.
Fuente: Elaboración propia.

En la matriz de correlaciones de Spearman (Figura 17), podemos observar que las variables *total_time*, *vms*, *rss* y *data* tienen una fuerte relación monótona positiva con la variable objetivo *total_time_target*.

En el caso del coeficiente de Kendall se "evalúa el grado de similitud entre dos conjuntos de rangos asignados a un mismo conjunto de objetos"[Abdi, 2007], en donde el coeficiente τ oscila en $[-1; 1]$ indicando que:

- Si $\tau > 0$, existe una correlación de orden positivo, esto indica que hay más pares de datos concordantes que discordantes, indicando una tendencia similar en el orden de los valores en ambas variables.
- Si $\tau < 0$, existe una correlación de orden negativo, esto indica que hay más pares de datos discordantes que concordantes, indicando una tendencia opuesta en el orden de los valores en ambas variables.

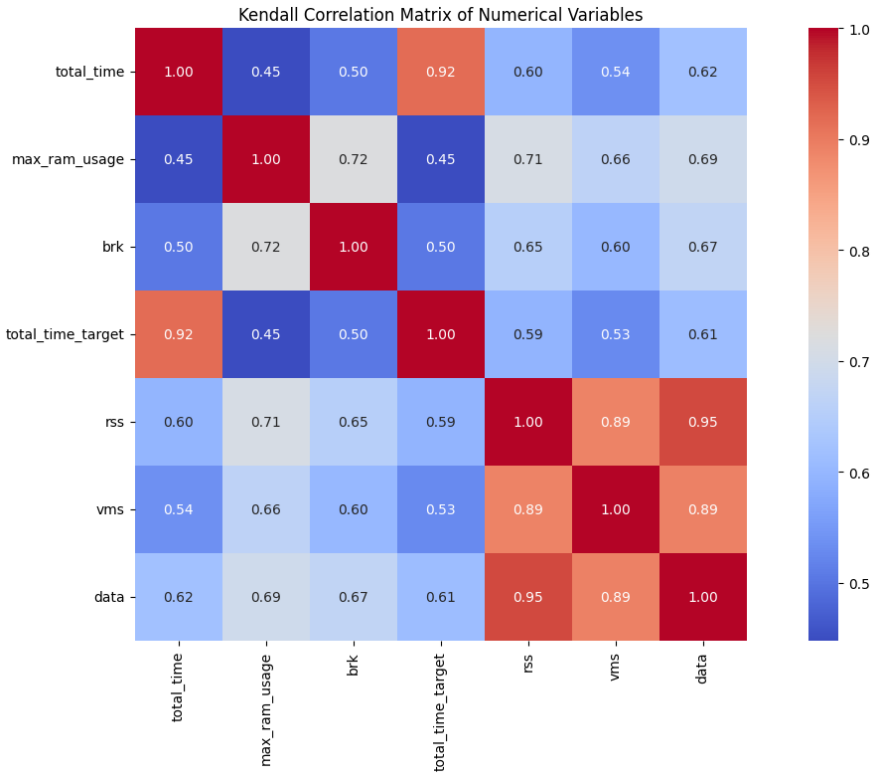


Figura 18: Matriz de correlación de Kendall. Solo se muestran las columnas con un coeficiente de correlación mayor a $0{,}4$ contra la variable objetivo.
Fuente: Elaboración propia.

Al analizar la matriz de correlación de Kendall (Figura 18), se observa que la variable *total_time* es la única que tiene una relación fuerte de orden positivo con respecto a la variable objetivo, similar a lo que se observó en el coeficiente de Pearson.

Con el análisis hecho a las distintas correlaciones, se tomará en cuenta un subconjunto de métricas, conformado por la unión de aquellas que mostraron un coeficiente fuerte en alguna de las correlaciones, para las variables de entrada de los modelos. En la sección 5.3 se realizará la comparación entre modelos que tienen acceso a todas las métricas y este subconjunto obtenido del análisis.

4.4. Creación de los conjuntos de entrenamiento y de prueba

Para la generación del conjunto de entrenamiento y de prueba, se realizaron las siguientes operaciones en el conjunto de datos:

1. Para realizar una comparativa válida con el estudio "*On the Fair Comparison of Optimization Algorithms in Different Machines*"[Arza *et al.*, 2023], se deben excluir del conjunto de prueba todos los registro que tengan los siguientes procesadores:
   - *Intel Core i5-12400F*, debido a que su puntuación *Passmark* es demasiado alta y no se encuentra dentro de los rangos con los que se calibró el modelo del estudio.
   - *Intel Core i5-1335U*, por no estar en el listado de puntuaciones *Passmark* utilizado en el estudio.
2. Añadir exclusivamente al conjunto de pruebas aquellos registros que cuenten con los procesadores *Intel Core i5-8300H* e *Intel Core i5-10300H*, tanto en la máquina base como en la máquina objetivo, para analizar el comportamiento de los modelos en computadores que no se encuentran en el conjunto de entrenamiento.
3. Añadir registros al conjunto de prueba mediante los siguientes pasos:
   a) Seleccionar aleatoriamente un tipo de prueba y un procesador objetivo (es decir, el procesador de la máquina objetivo).
   b) Añadir al conjunto de pruebas todos los registros que coincidan con el tipo de pruebas y el procesador objetivo seleccionados.
   c) Eliminar del conjunto de datos todos los registros que tengan el tipo de pruebas y procesador objetivo seleccionado, pero en su máquina base.
   d) Repetir hasta que el conjunto de pruebas tenga el $10\,\%$ del tamaño del conjunto de datos original.
4. El conjunto de entrenamiento se conformará por el resto de registros del conjunto de datos.

En la tabla 2 se observa la cantidad de datos trabajados.

Tabla 2: Cantidad de datos trabajados por conjunto.

Fuente: Elaboración Propia.

Conjunto	Cantidad de registros	Porcentaje
Original	2798	-
Self-join	72909	$100\,\%$
Entrenamiento	59165	$81\,\%$
Prueba	7368	$10\,\%$

4.5. Implementación de modelos predictivos

Se implementaron cuatro modelos de ML para la inferencia del tiempo total en la máquina objetivo, usando como entrada para todos los modelos las métricas descritas en el Anexo 7.1 y 7.2. Los modelos de ML usados fueron:

1. XGBoost
2. Feed-forward
3. Monte Carlo Dropout
4. TabNet

Para todos los modelos se implementó el siguiente preprocesado de los datos de entrada:

1. Se aplicó una transformación logarítmica a las métricas de tiempo total, tanto para la máquina base, como para la máquina objetivo.
2. Se normalizaron todas las métricas mediante un escalado `MinMax`, donde fueron transformadas dentro de un rango $[0; 1]$, ya que "la transformación de datos, como la normalización, puede mejorar la precisión y la eficiencia de los algoritmos"[Al Shalabi y Shaaban, 2006].

Se utilizaron las siguientes configuraciones para cada modelo.

4.5.1. XGBoost

Se utilizaron la mayoría de los parámetros en su configuración por defecto; solo se alteró la cantidad de rondas de *boosting* durante el entrenamiento, de 10 a 100 rondas. Además, se utilizó como métrica de evaluación la raíz del error cuadrático medio (RMSE) dado que es menos sensible al compararlo contra el error cuadrático medio (MSE) pero sigue penalizando errores grandes más que el error absoluto medio (MAE).

4.5.2. Feed-forward

La red *Feed-forward* propuesta para esta problemática consiste en 2 capas ocultas con funciones de activación *ReLU*, y se emplea la técnica de *dropout* para prevenir el *overfitting*. La capa de salida utiliza una función de activación *Softplus*, que asegura que las predicciones sean positivas.

El modelo se entrena con un *batch size* de 32 por 100 *epochs* utilizando el optimizador *Adam* con un *learning rate* de $0{,}0002$ y como función de pérdida se utilizó RMSE. Además, se implementa una estrategia de *early stopping* para evitar el *overfitting*, con una paciencia de 10 iteraciones.

4.5.3. Monte Carlo Dropout

La arquitectura propuesta incluye 3 capas ocultas, cada una con activaciones *ReLU* y regularizadores *dropout*. La capa de salida genera los parámetros de una distribución normal, permitiendo modelar la media y la incertidumbre de las predicciones.

El modelo se entrena con un *batch size* de 32 por 100 *epochs* utilizando el optimizador *Adam* con un *learning rate* de $0{,}0002$ y como se utilizó la función de perdida *Negative Log-Likehood* como función de perdida que evalúa la calidad de la cuantificación de incertidumbre del modelo. Además, se implementa una estrategia de *early stopping* para evitar el *overfitting*, con una paciencia de 25 iteraciones.

Además, para inferir el tiempo total objetivo de una entrada, se promedian 250 predicciones de la misma entrada.

4.5.4. TabNet

En TabNet se utilizaron todos sus parámetros en su configuración por defecto. Se utilizó como función objetivo RMSE.

CAPÍTULO 5
VALIDACIÓN DE LA SOLUCIÓN

En esta sección se analizarán los resultados en cuanto al rendimiento obtenido por los modelos, comparándolos con el modelo planteado en el estudio "*On the Fair Comparison of Optimization Algorithms in Different Machines*"[Arza *et al.*, 2023] con nombre `rtdhw`, en su calibración por defecto, así como con dos modelos de regresión lineal: uno que solo ocupará el tiempo de ejecución de la máquina base como entrada, y otro que, al igual que los demás modelos, empleará las demás métricas obtenidas en secciones anteriores.

5.1. Análisis de resultados

La métrica principal con la que se evaluará la precisión de los modelos es el error absoluto porcentual (MAPE), que se define como:

$$MAPE = 100\frac{1}{n}\sum_{i=1}^{n}\left|\frac{P_i - A_i}{A_i}\right|$$

Donde n es la cantidad de registros dentro del conjunto de pruebas, P_i es el valor predicho por el modelo sobre tiempo total de ejecución para la máquina objetivo para el registro i y A_i es el valor real del tiempo total para la máquina objetivo para el registro i.

Además para las comparaciones de MAPE se agregaron dos modelos:

- *ensemble*: compuesto por los modelos XGBoost, *Feed-forward* y TabNet.
- *ensemble_mcdropout*: que añade *Monte Carlo Dropout* al *ensemble* descrito anteriormente.

Los resultados conseguidos (Figura 19) indican que, a lo largo del conjunto de prueba, el modelo XGBoost consigue un desempeño superior en comparación con los demás modelos, seguido por los *ensembles* y la red *Feed-forward*.

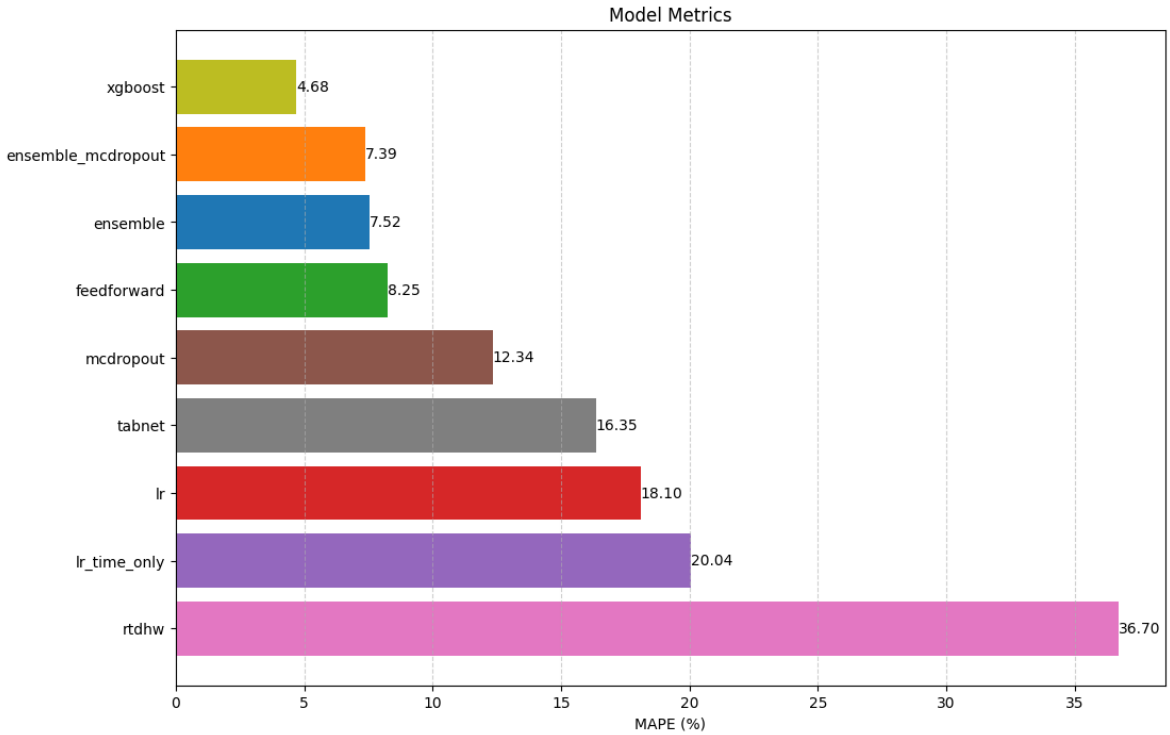


Figura 19: Análisis de MAPE generales. Mientras más bajo mejor.
Fuente: Elaboración propia.

Al analizar los resultados por rangos (Figura 20), se observa que XGBoost sigue siendo el mejor modelo en todos los rangos, a excepción del rango $[2{,}86; 5{,}08]$, donde la red *Feed-forward* presenta un mejor rendimiento.

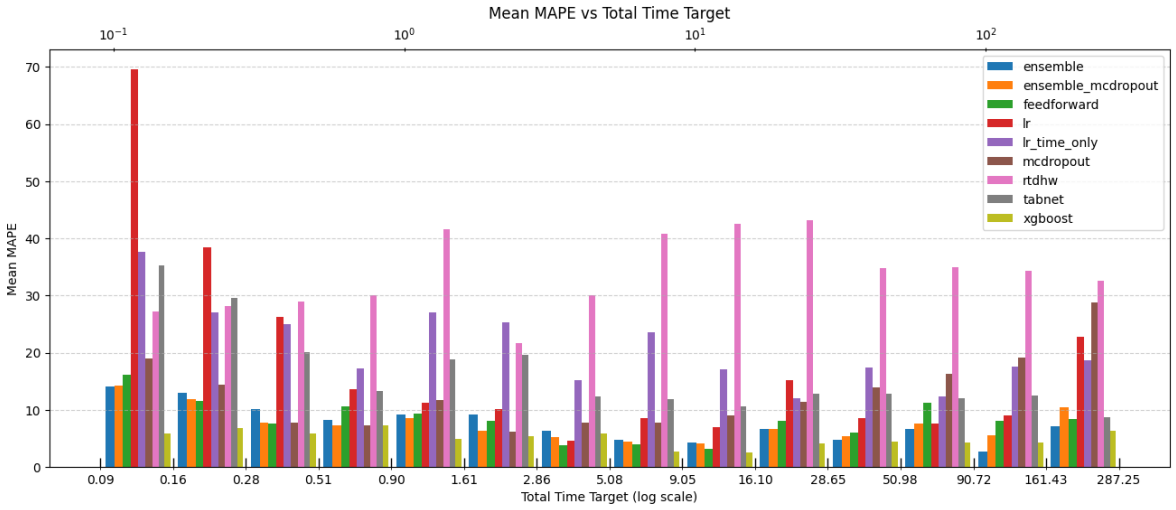


Figura 20: Análisis de MAPE por rangos. Mientras más bajo mejor.
Fuente: Elaboración propia.

Además, se observa (Figura 21) que el modelo *Monte Carlo Dropout* mantiene un rendimiento competitivo en tiempos menores a $5{,}08$, pero en tiempos mayores a $90{,}72$ presenta un error substancial.

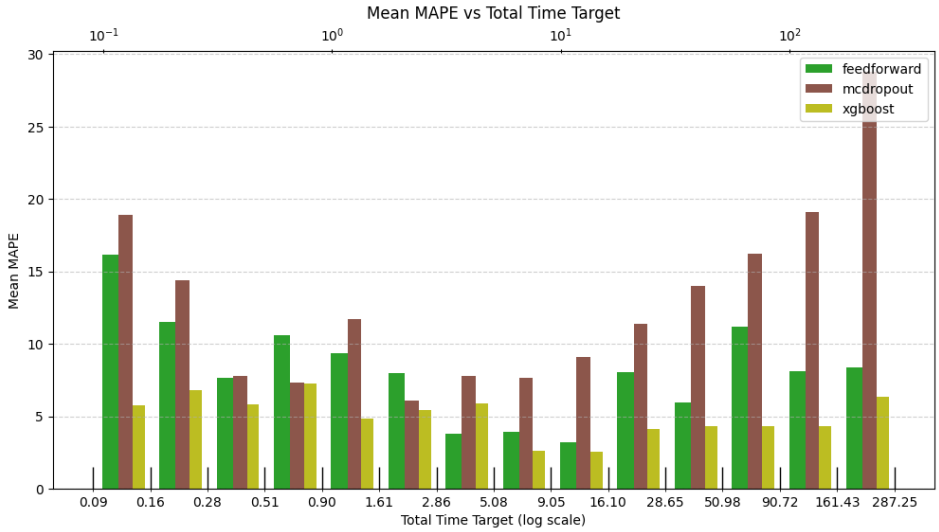


Figura 21: Análisis de MAPE por rangos (XGBoost, *Feed-forward* y *Monte Carlo Dropout*). Mientras más bajo mejor.

Fuente: Elaboración propia.

Al observar el comportamiento del modelo *Monte Carlo Dropout* (Figuras 22 y 23) a través del conjunto de datos, donde se realizaron 1000 inferencias para cada registro del conjunto de pruebas, la variabilidad de los intervalos de confianza en las predicciones para valores menores a $0{,}28$ y mayores a $28{,}65$ explica los MAPE obtenidos en la Figura 21.

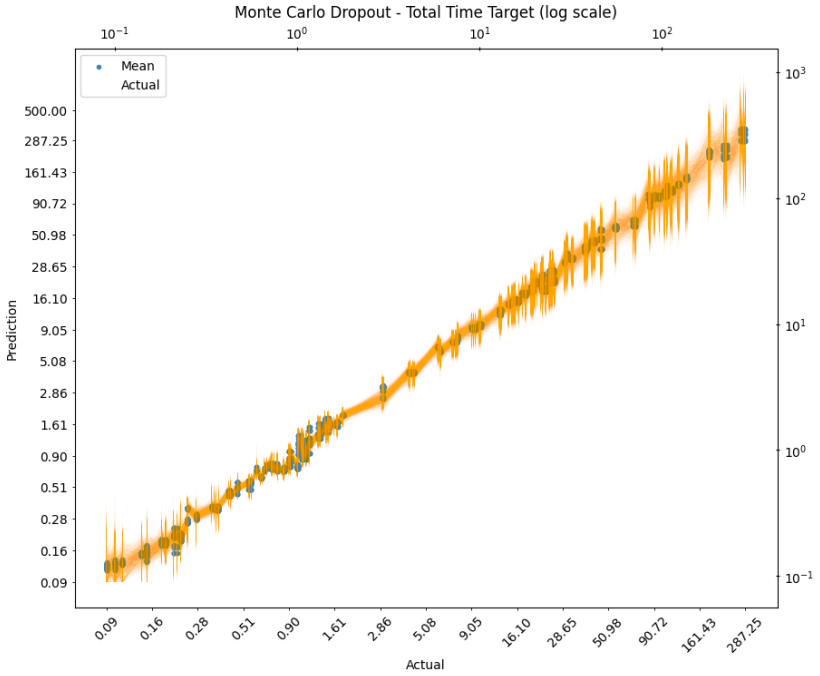


Figura 22: Distribución de predicciones de tiempo total de ejecución de modelo *Monte Carlo Dropout*.

Fuente: Elaboración propia.

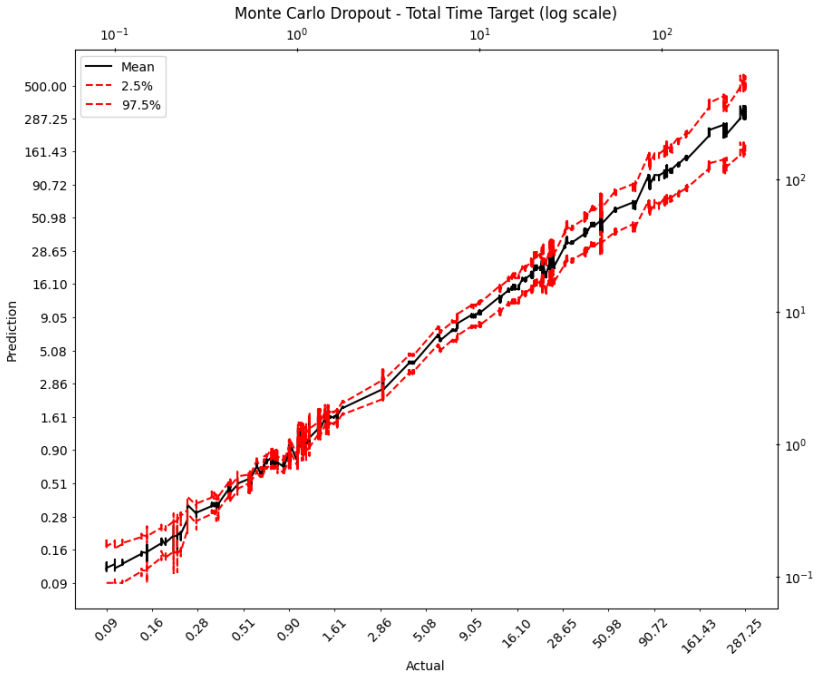


Figura 23: Distribución de predicciones de tiempo total de ejecución de modelo *Monte Carlo Dropout* con intervalos de confianza.

Fuente: Elaboración propia.

Al comparar (Figura 24) el modelo del estudio (`rtdhw`) con XGBoost y las regresiones lineales notamos que para tiempos totales mayores a $2{,}86$ la diferencia de MAPE entre `rtdhw` y los otros modelos es mayor del $10\,\%$, al compararlo solo con XGBoost se observa que para todos los rangos la diferencia de MAPE es mayor o igual al $15\,\%$.

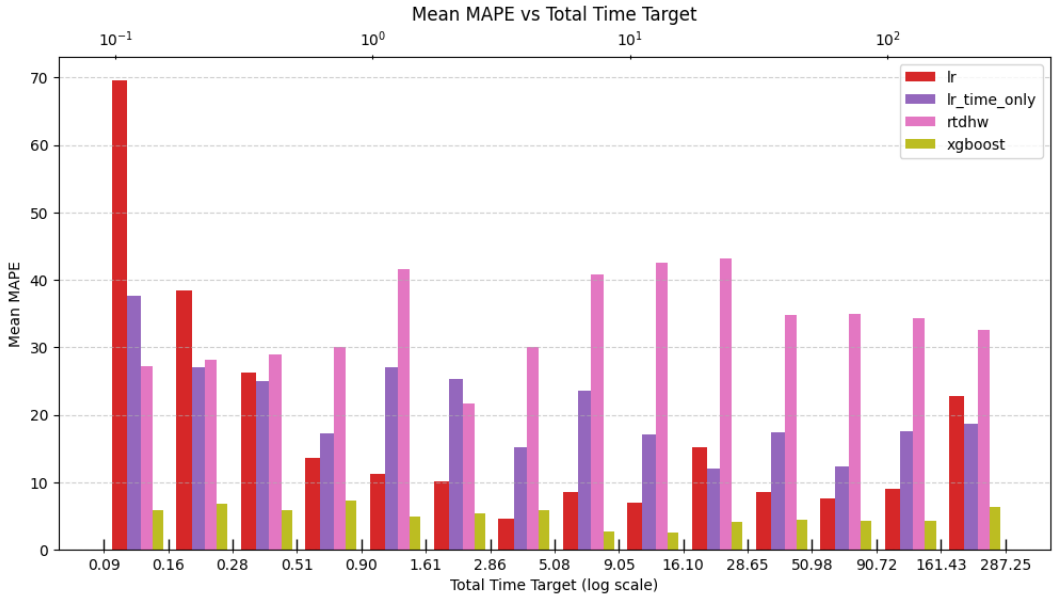


Figura 24: Análisis de MAPE rangos (XGBoost, regresión lineal, regresión lineal solo con tiempo total y `rtdhw`). Mientras más bajo mejor.
Fuente: Elaboración propia.

5.2. Análisis de resultados en subconjunto exclusivo de prueba

El subconjunto de prueba está conformado por todos los registros que tuvieron como origen los computadores: *Intel Core i5-8300H* y *Intel Core i5-10300H*.

Se puede destacar (Figura 25) que en el conjunto de prueba, el modelo TabNet es superado por la regresión lineal que utiliza solo el tiempo total de ejecución de la máquina objetivo como entrada. De manera similar al análisis con el conjunto de pruebas completo, el modelo XGBoost sigue siendo el que presenta un mejor rendimiento, seguido por los *ensembles* y el modelo *Feed-forward*. Al observar los resultados por rangos (Figura 26), se determina que estas conclusiones son válidas solo para el rango $[9{,}05; 28{,}65]$ y se necesitaría realizar pruebas con tiempos de ejecución menores y mayores para poder extrapolar.

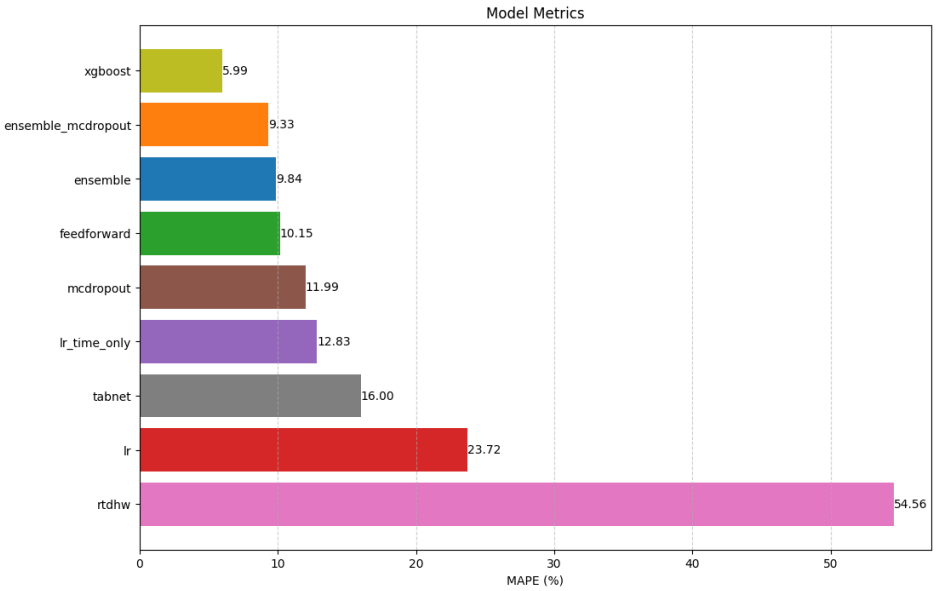


Figura 25: Análisis de MAPE con subconjunto de computadores exclusivos de prueba. Mientras más bajo mejor.

Fuente: Elaboración propia.

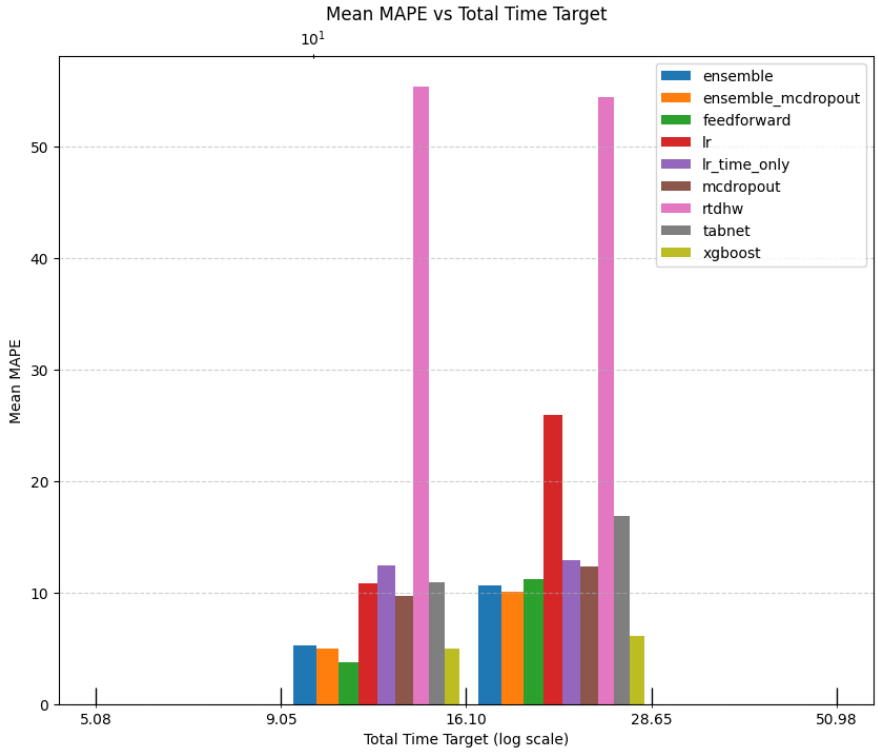


Figura 26: Análisis de MAPE por rangos con subconjunto de computadores exclusivos de prueba. Mientras más bajo mejor.

Fuente: Elaboración propia.

5.3. Comparación con subconjunto de métricas

El subconjunto esta conformado por las siguientes métricas:

- *total_time*
- *vms*
- *rss*
- *data*

Se obtuvo al unir todas las métricas que presentaban un coeficiente mayor a $|0{,}7|$ en alguna correlación en la sección 4.3.

Se observa que los resultados obtenidos (Figura 27) por los modelos entrenados con el subconjunto de métricas no logran superar el rendimiento de los modelos que tuvieron acceso a todas las métricas recolectadas.

Los modelos que solo ocupan el subconjunto de métricas tienen el sufijo `_corr`.

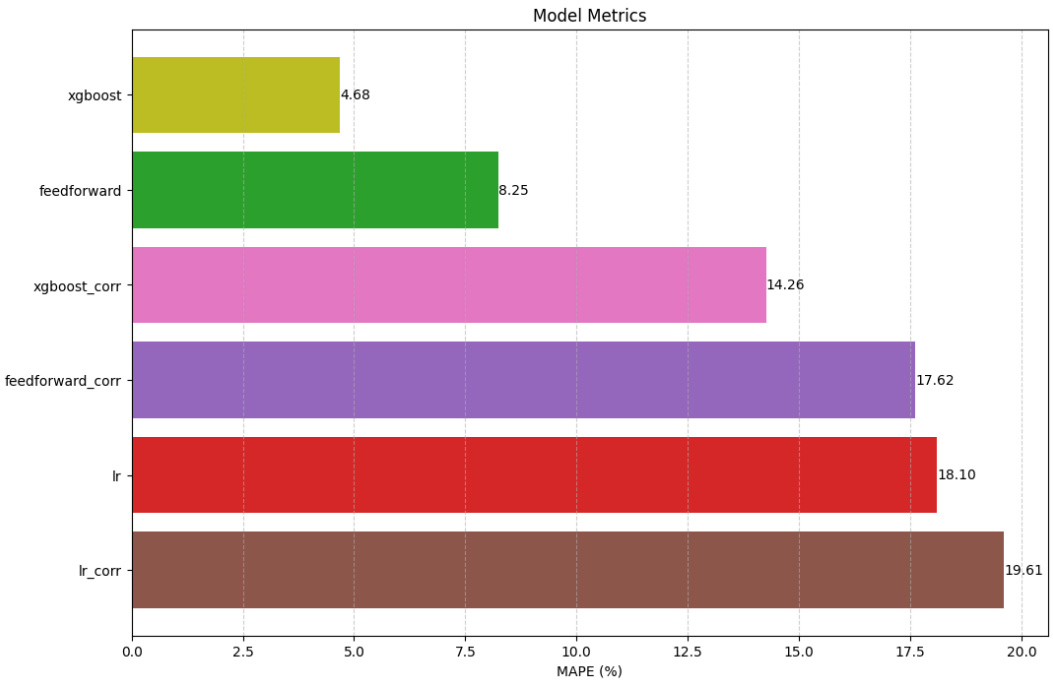


Figura 27: Análisis de MAPE con subconjunto de métricas de alta correlación. Mientras más bajo mejor.

Fuente: Elaboración propia.

Al realizar un análisis del MAPE por rangos (Figura 28), se observa que solo la regresión lineal se beneficia del uso del subconjunto de métricas para valores menores a $0{,}28$.

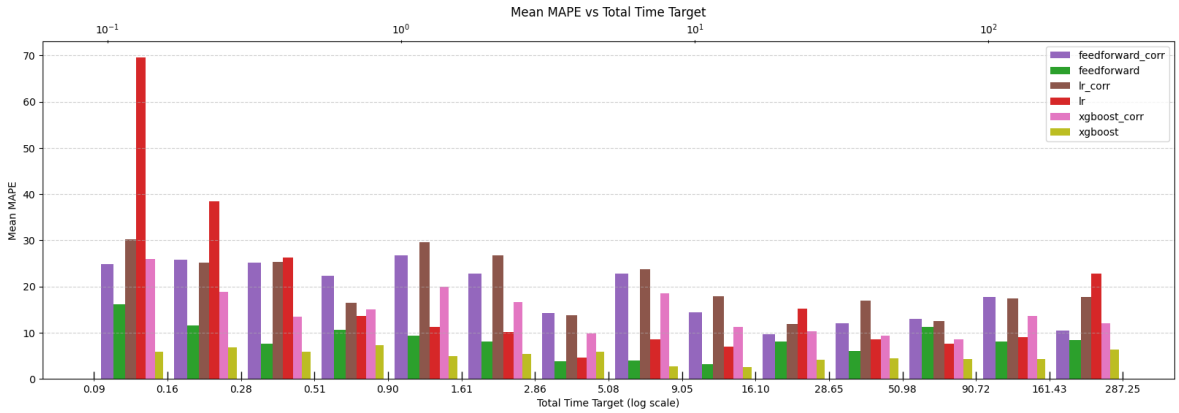


Figura 28: Análisis de MAPE por rangos con subconjunto de métricas de alta correlación. Mientras más bajo mejor.

Fuente: Elaboración propia.

5.4. Análisis SHAP

Se realizó el análisis SHAP para los modelos XGBoost y *Feed-forward*, ya que ambos eran los que presentaban mejores MAPE.

5.4.1. SHAP XGBoost

Al analizar la dispersión de valores de XGBoost (Figura 29), se observa que los valores altos de la variable tiempo total impactan[5] positivamente en la predicción del modelo, mientras que los valores bajos pueden impactar tanto positiva como negativamente. En cuanto a las demás variables, tienen una dispersión mínima, muy cercana a 0.

La magnitud del impacto de las variables (Figura 30) confirma que el tiempo total es la variable más importante en la predicción, mientras que las demás variables tienen un impacto muy leve en el resultado.

[5]El impacto se refiere a la influencia o magnitud que una variable tiene en la predicción del modelo. Un impacto negativo contribuye a que el modelo prediga un valor más bajo que el promedio, mientras que un impacto positivo contribuye a un valor más alto que el promedio.

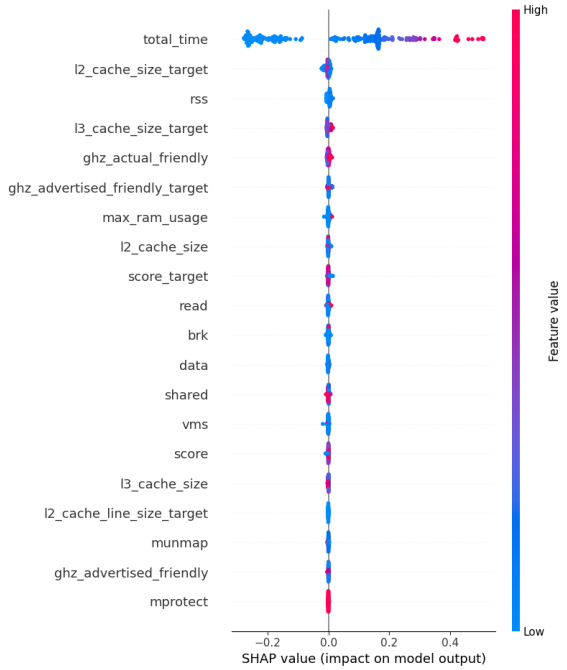


Figura 29: Explicación global de modelo XGBoost según magnitud y dispersión de *Shapley values*.

Fuente: Elaboración propia.

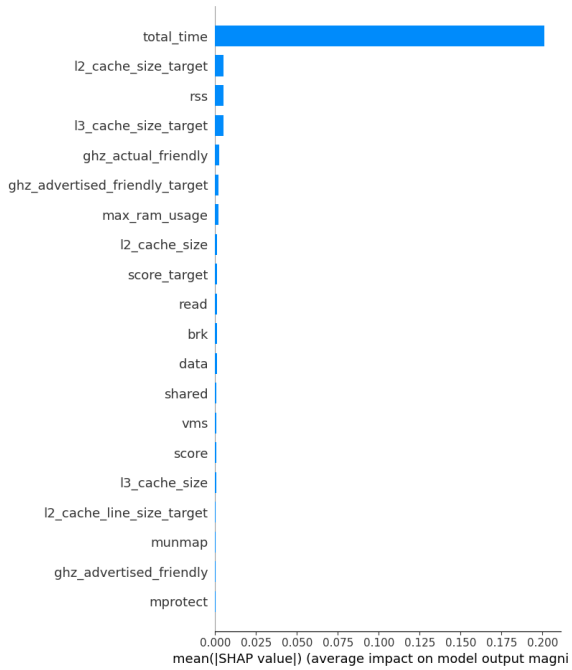


Figura 30: Explicación global de modelo XGBoost según magnitud absoluta de *Shapley values*.

Fuente: Elaboración propia.

En la Figura 31 se observa un cambio en la importancia de las variables a medida que disminuye el valor de la predicción del tiempo total de ejecución de la máquina objetivo. Para valores del orden de 10^2 el tiempo total (de la máquina base) tiene un gran impacto positivo en las predicciones del modelo, pero transiciona a un impacto negativo al acercarse a valores del orden de 10^{-1}.

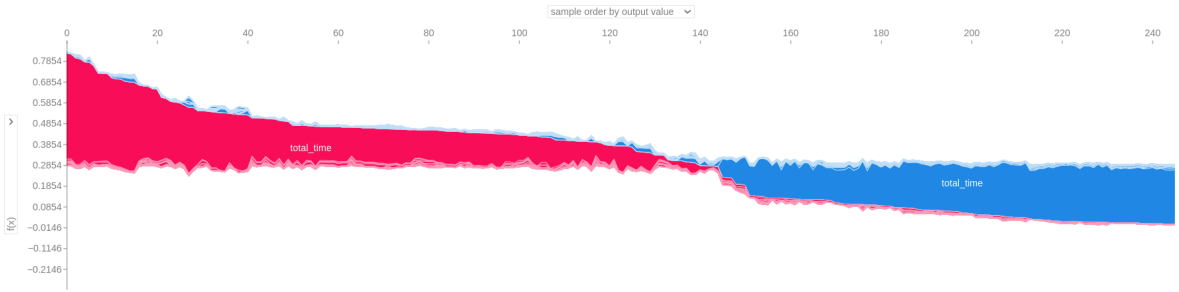


Figura 31: Gráfico SHAP de fuerza apilado de modelo XGBoost. Valores de tiempo total objetivo ordenados de mayor a menor de izquierda a derecha respectivamente. Un color rojo indica un impacto positivo y un color azul a un impacto negativo.

Fuente: Elaboración propia.

A continuación se presentan tres instancias (con orden 10^2, 10^1 y 10^{-1} respectivamente) que

ejemplifican el cambio de importancia de la Figura 31:

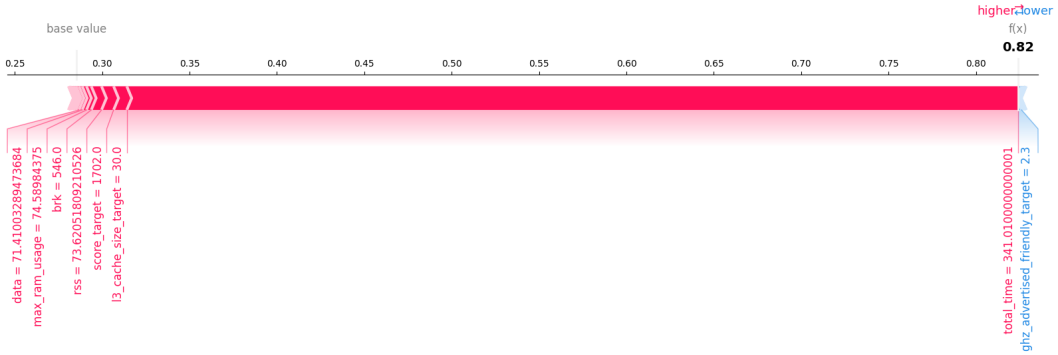


Figura 32: Gráfico SHAP de fuerza, instancia 1°, Predicción : $274{,}89$ | Real : $269{,}24$
Fuente: Elaboración propia.

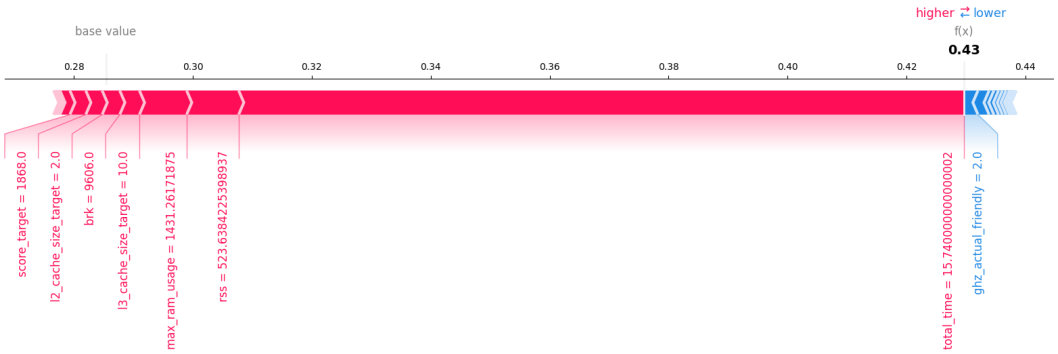


Figura 33: Gráfico SHAP de fuerza, instancia 2°, Predicción : $18{,}53$ | Real : $17{,}86$
Fuente: Elaboración propia.

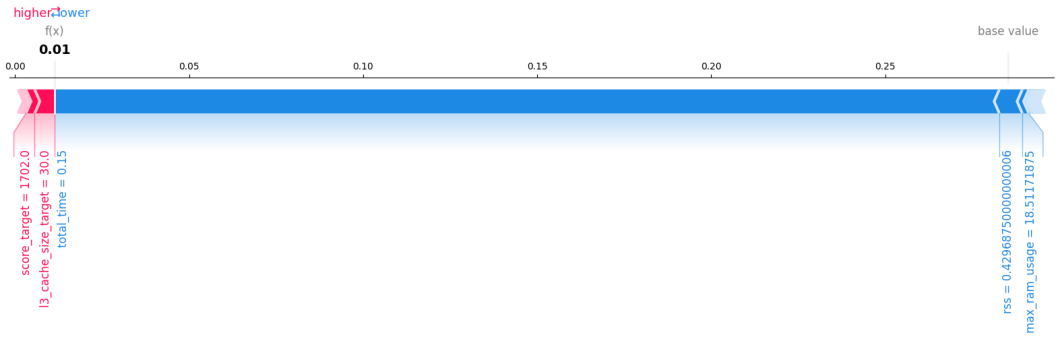


Figura 34: Gráfico SHAP de fuerza, instancia 3°, Predicción : $0{,}18$ | Real : $0{,}19$
Fuente: Elaboración propia.

5.4.2. SHAP Feed-forward

En la dispersión de valores de *Feed-forward* (Figura 35), similar con a lo observado en XGBoost, que los valores altos de la variable tiempo total impactan positivamente en la predicción del modelo, mientras que los valores bajos pueden impactar tanto positiva como negativamente. Sin embargo, las demás variables presentan una dispersión pequeña, pero mayor a lo observado en XGBoost, lo que indica que variables como *count*, *l2_cache_associativity* y *stat* tienen un poco más de importancia.

La magnitud del impacto de las variables (Figura 36) deja en evidencia que el tiempo total es la variable más importante en la predicción, aun así, cabe recalcar que tomando en cuenta la magnitud de las demás métricas, se puede determinar que el modelo *Feed-forward* toma en cuenta más métricas que el modelo *XGBoost*.

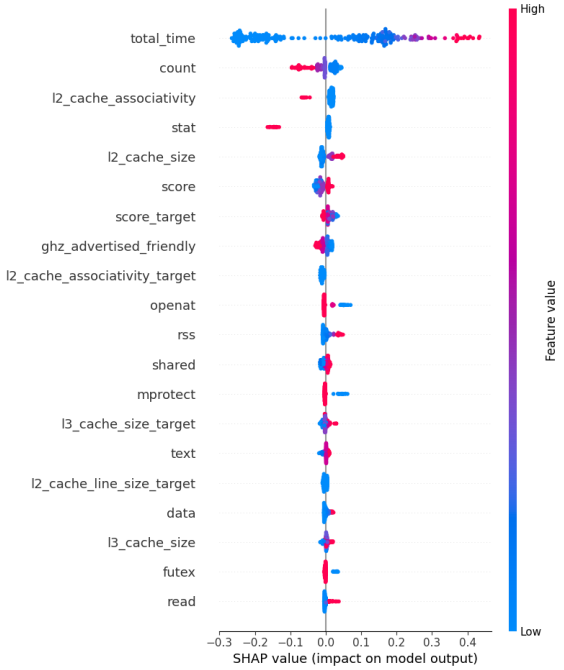


Figura 35: Explicación global de modelo *Feed-forward* según magnitud y dispersión de *Shapley values*.
Fuente: Elaboración propia.

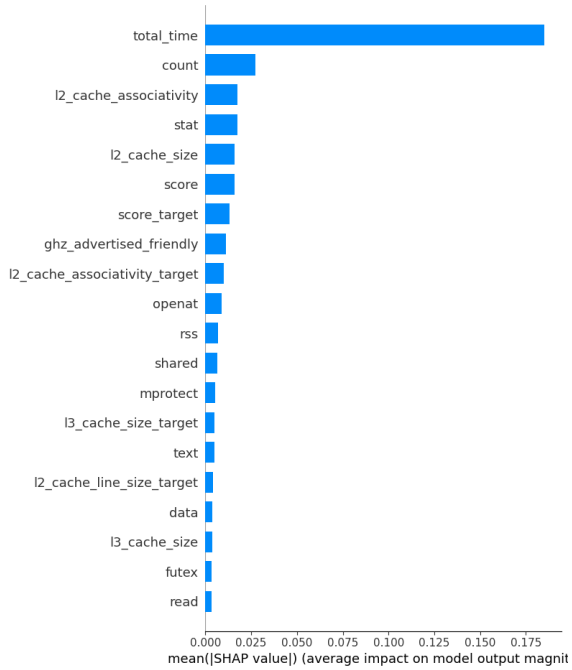


Figura 36: Explicación global de modelo *Feed-forward* según magnitud absoluta de *Shapley values*.
Fuente: Elaboración propia.

Al igual que con XGBoost, se observa un cambio en la importancia de las variables (Figura 37) a medida que disminuye el valor de la predicción del tiempo total de ejecución de la máquina objetivo. Para valores del orden de 10^2 el tiempo total (de la máquina base) tiene un gran impacto positivo en las predicciones del modelo, pero transiciona a un impacto negativo al acercarse a valores del orden de 10^{-1}. Aunque, en este caso, se puede destacar que otras variables tienen un ligero aumento de importancia a lo largo del conjunto de datos.

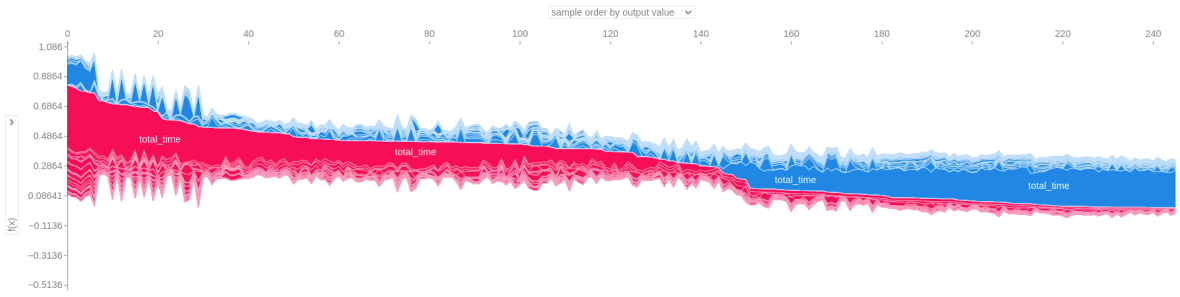


Figura 37: Gráfico SHAP de fuerza apilado de modelo *Feed-forward*. Valores de tiempo total objetivo ordenados de mayor a menor de izquierda a derecha respectivamente. Un color rojo indica un impacto positivo y un color azul a un impacto negativo.
Fuente: Elaboración propia.

A continuación se presentan tres instancias (con orden 10^2, 10^1 y 10^0 respectivamente) que ejemplifican el cambio de importancia de la Figura 37:

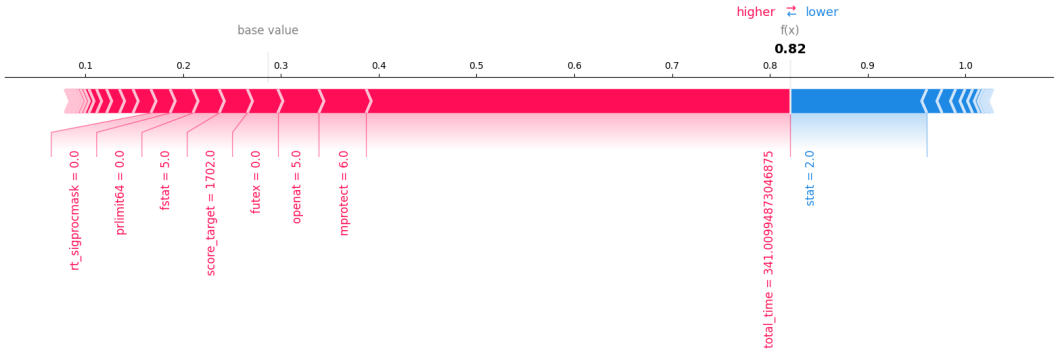


Figura 38: Gráfico SHAP de fuerza, instancia 1°, Predicción : $269{,}20$ | Real : $269{,}24$
Fuente: Elaboración propia.

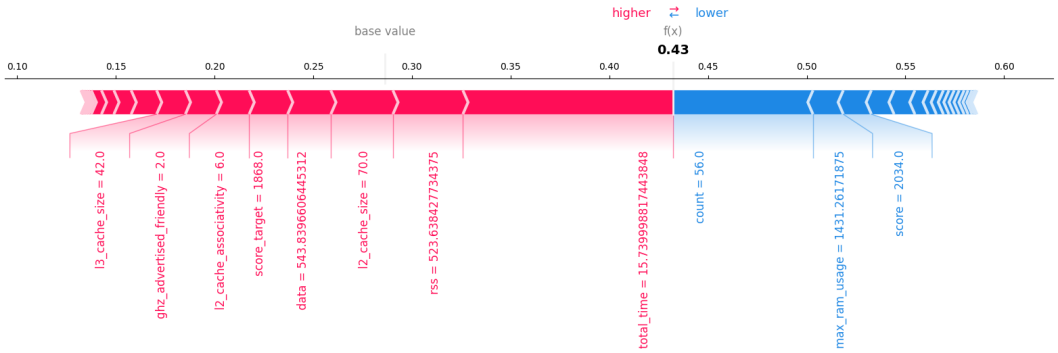


Figura 39: Gráfico SHAP de fuerza, instancia 2°, Predicción : $18{,}86$ | Real : $17{,}86$
Fuente: Elaboración propia.

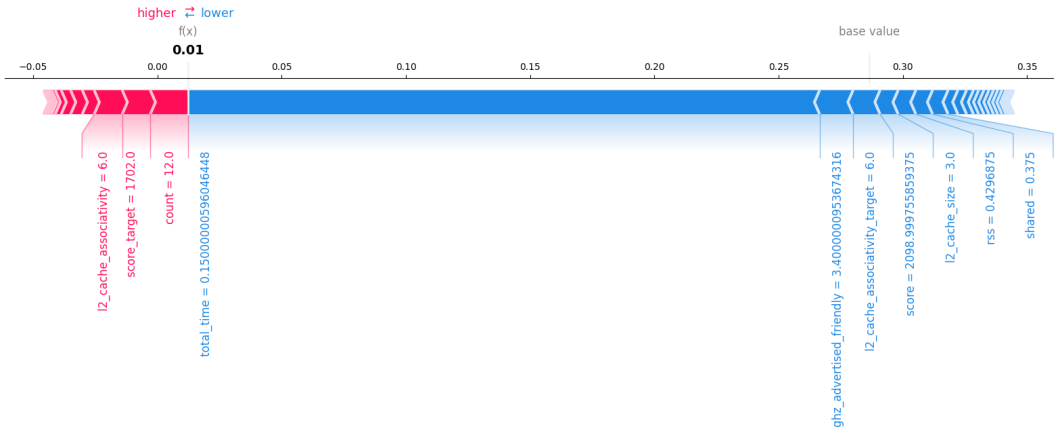


Figura 40: Gráfico SHAP de fuerza, instancia 3°, Predicción : $0{,}185$ | Real : $0{,}19$
Fuente: Elaboración propia.

CAPÍTULO 6
CONCLUSIONES

6.1. Conclusiones generales

A lo largo de esta memoria, se ha abordado el problema de predecir el tiempo de ejecución de algoritmos en una máquina objetivo, utilizando las métricas de consumo de recursos obtenidas de una máquina base, especificaciones de *hardware* y conteo de llamadas del sistema.

Para ello, se creó un conjunto de datos diverso y representativo mediante la ejecución de distintas instancias de problemas de optimización combinatoria como TSP, KNP y N-Queens en diferentes computadores facilitados la Universidad de California, Berkeley mediante el *cluster* de computación de alto desempeño *Savio*, además de un conjunto de computadores de sobremesa y *notebooks* personales gracias a la ayuda de colegas de la universidad. Para mantener un ambiente de prueba homogéneo y controlado a través de todos los sistemas, se creó un *pipeline* automatizado en `Python` que se ejecutaría en contenedores gracias a *Docker* y *Apptainer*.

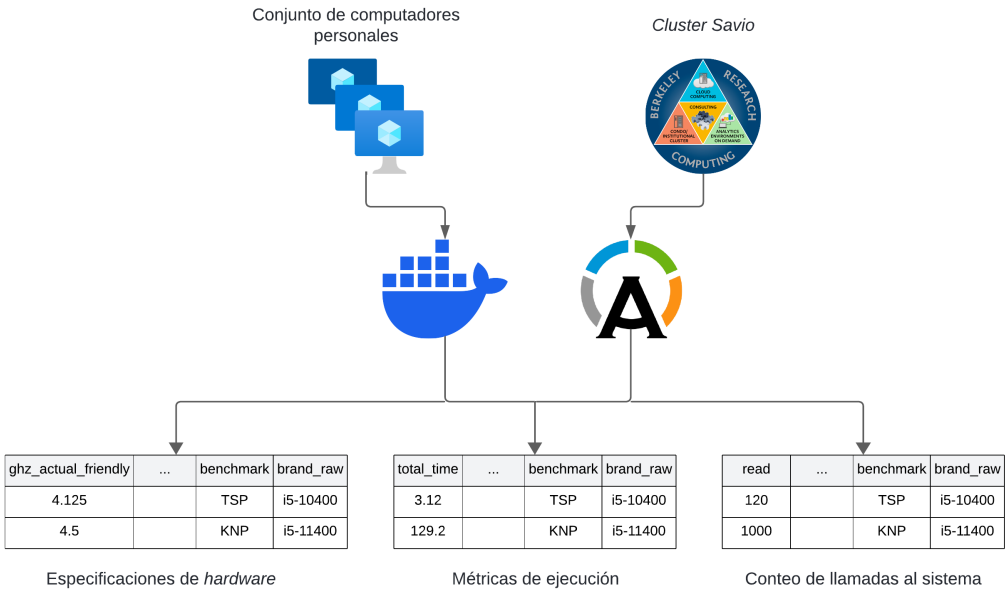


Figura 41: Proceso de obtención de datos.
Fuente: Elaboración propia, en *Lucidchart*, iconos propiedad de *Docker*, *Apptainer* y Universidad de Berkeley.

Luego de operaciones como estandarización de los datos y unión de fuentes como métricas de ejecución, especificaciones de *hardware* y conteo de llamadas al sistema, se creó un conjunto de datos al que se le realizó un análisis exploratorio, donde se pudo determinar que la escala logarítmica era la mejor representación para el conjunto de datos utilizado. Al utilizar los coeficientes de correlación de Pearson, Spearman y Kendall se pudieron observar ciertas relaciones, que si bien no llegaron a generar mejores resultados, sirven como base para estudios futuros.

Se generaron un conjunto de datos de entrenamiento y un conjunto de datos de prueba mediante operaciones de normalización, para que los modelos de ML propuestos, como XGBoost, redes neuronales *Feed-forward*, *Monte Carlo Dropout* y TabNet, fueron implementados y probados con estos conjuntos.

Finalmente, se analizaron los resultados obtenidos por los modelos, donde XGBoost presentó el mejor desempeño según la evaluación de la métrica MAPE, seguido de la red *Feed-forward*.

Se implementaron dos *ensembles* que mostraron un rendimiento competitivo, a la par de la red *Feed-forward*. También se analizó el comportamiento de la red bayesiana *Monte Carlo Dropout* mediante los estadísticos media e intervalos de confianza, obtenidos luego de reiteradas inferencias. Además, se realizaron explicaciones SHAP en las dos redes que, individualmente, presentaron mejor desempeño (XGBoost y *Feed-forward*), donde se llegó a la conclusión de que *Feed-forward* daba más importancia que XGBoost, a otras variables que no fueran el tiempo de ejecución de la máquina base.

6.2. Cumplimiento de objetivos

El resultado de esta memoria se puede evaluar en función del cumplimiento de los objetivos específicos definidos en la sección 1.2.2, ya que cumplir con estos objetivos implica el cumplimiento del objetivo general.

Los objetivos específicos son:

O1) **Elaborar un conjunto de datos diverso y representativo para entrenar el modelo de ML, basado en las estadísticas del consumo de recursos de varios algoritmos en diferentes máquinas.**

La obtención de datos en la sección 4.1 asegura que el conjunto de datos creado posteriormente en la sección 4.2 sea diverso y representativo, mediante la ejecución de 92 pruebas en 8 computadores pertenecientes al *cluster* de computación de alto desempeño *Savio* y otros 6 computadores de sobremesa y *notebooks* personales, cada uno con distintas especificaciones. Esto permitió obtener 2798 registros, que, luego de aplicar las transformaciones para adaptarlo a la problemática de predicción del tiempo

total de ejecución en una máquina objetivo, resultó en un conjunto de datos de 72909 registros. Por lo tanto, este objetivo se logró cumplir.

O2) **Evaluar la calidad del modelo de ML aplicando métricas estándar (MSE, MAE, R^2).**

En la sección 5.1, se realizó la evaluación de múltiples modelos usando la métrica estándar MAPE, que es el equivalente porcentual del MAE y se usa para medir el error de forma consistente en modelos de predicción donde la escala de los valores predichos puede variar significativamente, como se descubrió en el análisis exploratorio (Sección 4.3). Al usar una métrica estándar como MAPE, se cumple el objetivo de aplicar métricas estándar para evaluar la calidad de los múltiples modelos de ML implementados y determinar cuál tiene el menor error, como es el caso de XGBoost.

O3) **Comparar los resultados de la solución propuesta con los resultados del estado del arte.**

Se implementaron cuatro modelos de ML en la sección 4.5, que hacían uso de todas las métricas obtenidas (XGBoost, *Feed-forward*, *Monte Carlo Dropout* y TabNet) y se llevó a cabo la comparación con el modelo de estimación `rtdhw` del estudio "*On the Fair Comparison of Optimization Algorithms in Different Machines*"[Arza *et al.*, 2023] en la sección 5.1, que "dado el tiempo de ejecución de un algoritmo en una máquina, estima el tiempo de ejecución del mismo algoritmo en otra máquina"[Arza *et al.*, 2023]. Todos los modelos implementados lograron superar al modelo `rtdhw` en el conjunto de pruebas establecido, tomando como métrica de evaluación el MAPE. Al realizar esta comparación, se cumple el objetivo de comparar resultados con el estado del arte.

O4) **Comunicar a la comunidad académica los resultados de esta investigación.**

Al realizar la publicación y defensa de esta memoria en el ámbito académico del trabajo de título (INF-310), se realizaría el primer paso a comunicar a la comunidad académica los resultados de esta investigación, pasos futuros consistirían en publicar este trabajo en una conferencia. Con esto establecido, último objetivo específico pasa a continuo y da paso al trabajo futuro.

Dado que se cumplieron los objetivos específicos O1), O2) y O3), y el primer paso del O4) se establece como esta publicación, se puede declarar que el objetivo general, que consiste en: **Construir un modelo de ML que prediga el consumo de recursos computacionales en una máquina objetivo a partir del consumo de recursos y características una máquina base**. Se ha cumplido, al implementar, no uno, sino cuatro modelos capaces de predecir una de las principales variables que afectan el consumo de recursos, el tiempo total de ejecución de los programas.

6.3. Trabajo futuro

Como se mencionó en la sección anterior, parte del trabajo futuro y objetivo continuo consiste en llevar este estudio a una conferencia y conseguir su publicación, para lograr comunicar

a más académicos sobre los resultados conseguidos en el estudio.

También se puede destacar como parte del trabajo futuro la expansión del conjunto de datos, ya sea añadiendo más computadores con diferentes características o creando más pruebas que permitan obtener más registros con un tiempo total de ejecución más cercano al comportamiento actual de los programas en la industria de computación de alto desempeño o incluso en la nube. Además, se podría expandir la capacidad de predicción para incluir programas que sean *multi-threading*. Otros posibles temas para un trabajo de título podría generarse a partir de una de estas áreas, donde al extender la cantidad de métricas predichas, recuperar más métricas que pudieran ser útiles (especialmente aquellas bloqueadas detrás de los permisos de superusuario), utilizar otros tipos de modelos del estado del arte o implementar estas predicciones en algún tipo de *scheduler*, se podría lograr un gran avance en el ámbito de la predicción de consumo de recursos computacionales.

Finalmente, otro avance logrado en esta memoria es la construcción de una API que permite acceder a las inferencias de los modelos presentados. Actualmente, la API se encuentra funcional y disponible en el repositorio del proyecto. Como parte del trabajo a futuro, se podría implementar en un servidor remoto para que cualquiera pueda realizar inferencias o descargar los pesos de los modelos y poder experimentar localmente con ellos.

6.4. Palabras finales del autor

Al escribir esta memoria, se presentaron interesantes desafíos, desde la investigación previa necesaria para identificar qué métricas utilizar o a cuáles se tenía acceso, hasta la implementación de la extracción de datos. Muchas métricas requerían permisos elevados, como es el caso de `perf`, lo que impuso restricciones al recopilar los datos. Además, al revisar la literatura disponible sobre la problemática, no se encontró ningún conjunto de datos viable para la investigación que combinara tiempos de ejecución y perfilados del sistema. Por lo tanto, se decidió crear un conjunto de datos desde cero, y gracias a mi Profesor guía Roberto Asín y a la Universidad de California, Berkeley, que facilitó el acceso a su *cluster*, también a colegas que cedieron sus computadores para ejecutar las pruebas, de los cuales se pudieron obtener bastantes registros.

Al momento de implementar las redes neuronales, me impuse un reto personal: aprender `PyTorch`, ya que hasta ese momento solo había utilizado un poco `TensorFlow`. Aunque la curva de aprendizaje fue complicada, se pudo superar. Incluso hasta el último momento, continué aprendiendo cosas nuevas y quedé encantado con la forma en que se programa usando este *framework*.

Extiendo la invitación a todos los estudiantes memoristas a que sigan lo que les apasiona. En esta memoria, tuve que crear desde cero mi propio conjunto de datos con *pipelines* que los generaran, aprender `PyTorch`, y cada reto fue una experiencia enriquecedora que contribuyó a mi crecimiento como profesional.

CAPÍTULO 7
ANEXOS

7.1. Descripciones de métricas recolectadas

Métricas de ejecución

- *total_time*: Tiempo de ejecución del programa en segundos.
- *total_cpu_usage*: Uso total porcentual del procesador.
- *max_ram_usage*: Cantidad máxima de RAM usada en kilobytes.
- *rss*: Cantidad de memoría física no intercambiada utilizada por un proceso (*Resident Set Size*).
- *vms*: Cantidad total de memoria virtual utilizada por el proceso (*Virtual Memory Size*).
- *shared*: Memoria que podría ser potencialmente compartida con otros procesos.
- *text*: Cantidad de memoria dedicada al código ejecutable (*Text Resident Set*).
- *data*: Cantidad de memoria física dedicada a elementos que no son código ejecutable (*data resident set*).

Especificaciones de *hardware*

- *count*: Número de *threads* del procesador.
- *l2_cache_size*: Tamaño de la caché L2 en megabytes.
- *l3_cache_size*: Tamaño de la caché L3 en megabytes.
- *l2_line_size*: Tamaño de la línea de caché L2, indica cuántos bytes se pueden transferir en un acceso de caché.
- *l2_cache_associativity*: Grado de asociatividad de la caché L2.
- *ghz_actual_friendly*: Frecuencia del procesador actualmente en gigahercios.
- *ghz_advertised_friendly*: Frecuencia del procesador comercialmente en gigahercios.
- *score*: Puntuación *single-thread* del procesador en *Passmark*.

Llamadas al sistema (X86_64)

- *close*: Cierra un descriptor de archivo, liberando los recursos asociados con él.
- *brk*: Cambia el segmento de datos del proceso para incrementar o disminuir la cantidad de memoria asignada al programa en tiempo de ejecución.
- *prlimit64*: Obtiene o establece los límites de recursos para un proceso.
- *set_tid_address*: Establece la dirección donde se almacenará el ID del *thread* para un proceso.
- *futex*: Es método para esperar hasta que cierta condición se vuelve verdadera.
- *read*: Intenta leer datos desde un descriptor de archivo hacia un *buffer*.
- *mmap*: Crea nuevas asociaciones en la memoria virtual del proceso.
- *fstat*: Retorna información acerca de un archivo.
- *set_robust_list*: Establece una lista de estructuras de datos que deben ser limpiadas automáticamente si un *thread* termina inesperadamente.
- *openat*: Abre un archivo especificado por un *pathname*.
- *rt_sigprocmask*: Manipula la máscara de señales del proceso.
- *rt_sigaction*: Cambia las acciones tomadas por el proceso cuando recibe una señal.
- *stat*: Retorna información del archivo, sin requerir que esté abierto previamente.
- *mprotect*: Cambia las protecciones de memoria de un área en el espacio de direcciones del proceso.
- *munmap*: Desasocia un área de memoria previamente asociada con *mmap*.

7.2. Descripciones de métricas de la máquina objetivo

- ***total_time_target***[6]: Tiempo de ejecución del programa en la máquina objetivo en segundos.
- *l2_cache_size_target*: Tamaño de la caché L2 en megabytes de la máquina objetivo.
- *l3_cache_size_target*: Tamaño de la caché L3 en megabytes de la máquina objetivo.

[6]Variable objetivo a predecir.

- *l2_line_size_target*: Tamaño de la línea de caché L2 de la máquina objetivo, indica cuántos bytes se pueden transferir en un acceso de caché.
- *l2_cache_associativity_target*: Grado de asociatividad de la caché L2 de la máquina objetivo.
- *ghz_advertised_friendly_target*: Frecuencia del procesador comercialmente de la máquina objetivo en gigahercios.
- *score_target*: Puntuación *single-thread* del procesador de la máquina objetivo en *Passmark*.

7.3. Especificaciones de instancias savio usadas

Tabla 3: Especificaciones del conjunto de computadores del *cluster* de computación de alto rendimiento *Savio* usado en el estudio.

Fuente: Elaboración Propia.

Partición	CPU	RAM [GB]	Puntuación Passmark *single thread* 2021
savio2	*Intel Xeon E5-2670 v3*	64	1702
savio2	*Intel Xeon E5-2650 v4*	64	1507
savio2	*Intel Xeon E5-2680 v4*	64	1952
savio2_htc	*Intel Xeon E5-2643 v3*	128	2099
savio2_gpu	*Intel Xeon E5-2623 v3*	64	1868
savio3	*Intel Xeon Gold 6130*	96	1880
savio3	*Intel Xeon Gold 6230*	96	2285
savio4_htc	*Intel Xeon Gold 6330*	256	2035

7.4. Especificaciones de computadores personales usados

Tabla 4: Especificaciones del conjunto de computadores personales usado en el estudio.

Fuente: Elaboración Propia.

Tipo	CPU	RAM [GB]	Puntuación Passmark *single thread* 2021
Escritorio	*Intel Core i5-10400*	16	2589
Escritorio	*Intel Core i5-12400F*	32	3511
Portátil	*Intel Core i5-8300H*	8	2319
Portátil	*Intel Core i5-10300H*	8	2630
Portátil	*Intel Core i5-1135G7*	8	2731
Portátil	*Intel Core i5-1335U*	8	-

REFERENCIAS BIBLIOGRÁFICAS

[Abdi, 2007] Abdi, H. (2007). The kendall rank correlation coefficient. *Encyclopedia of measurement and statistics*, 2:508–510.

[Abraham, 2005] Abraham, A. (2005). Artificial neural networks. *Handbook of measuring system design*.

[Agatonovic-Kustrin y Beresford, 2000] Agatonovic-Kustrin, S. y Beresford, R. (2000). Basic concepts of artificial neural network (ann) modeling and its application in pharmaceutical research. *Journal of pharmaceutical and biomedical analysis*, 22(5):717–727.

[Al Shalabi y Shaaban, 2006] Al Shalabi, L. y Shaaban, Z. (2006). Normalization as a preprocessing engine for data mining and the approach of preference matrix. En *2006 International conference on dependability of computer systems*, pp. 207–214. IEEE.

[Arik y Pfister, 2021] Arik, S. Ö. y Pfister, T. (2021). Tabnet: Attentive interpretable tabular learning. En *Proceedings of the AAAI conference on artificial intelligence*, volumen 35, pp. 6679–6687.

[Arza *et al.*, 2023] Arza, E., Ceberio, J., Irurozki, E., y Pérez, A. (2023). On the fair comparison of optimization algorithms in different machines. *arXiv preprint arXiv:2305.07345*.

[Badillo *et al.*, 2020] Badillo, S., Banfai, B., Birzele, F., Davydov, I. I., Hutchinson, L., Kam-Thong, T., Siebourg-Polster, J., Steiert, B., y Zhang, J. D. (2020). An introduction to machine learning. *Clinical pharmacology & therapeutics*, 107(4):871–885.

[Bebis y Georgiopoulos, 1994] Bebis, G. y Georgiopoulos, M. (1994). Feed-forward neural networks. *Ieee Potentials*, 13(4):27–31.

[Bielecki y Śmiałek, 2023] Bielecki, J. y Śmiałek, M. (2023). Estimation of execution time for computing tasks. *Cluster Computing*, 26(6):3943–3956.

[Chen y Guestrin, 2016] Chen, T. y Guestrin, C. (2016). Xgboost: A scalable tree boosting system. En *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794.

[Chiueh y Brook, 2005] Chiueh, S. N. T.-c. y Brook, S. (2005). A survey on virtualization technologies. *Rpe Report*, 142.

[Conover, 1999] Conover, W. J. (1999). *Practical nonparametric statistics*, volumen 350. john wiley & sons.

[Dittakavi, 2021] Dittakavi, R. S. S. (2021). Deep learning-based prediction of cpu and memory consumption for cost-efficient cloud resource allocation. *Sage Science Review of Applied Machine Learning*, 4(1):45–58.

[Docker, 2020] Docker, I. (2020). Docker. *Inea].[Junio de 2017]. Disponible en: https://www. docker. com/what-docker*.

[Drucker *et al.*, 1996] Drucker, H., Burges, C. J., Kaufman, L., Smola, A., y Vapnik, V. (1996). Support vector regression machines. *Advances in neural information processing systems*, 9.

[Friedman, 2001] Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pp. 1189–1232.

[Gal y Ghahramani, 2016] Gal, Y. y Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. En *international conference on machine learning*, pp. 1050–1059. PMLR.

[Goyal, 2010] Goyal, S. (2010). A survey on travelling salesman problem. En *Midwest instruction and computing symposium*, pp. 1–9.

[Gupta *et al.*, 2008] Gupta, C., Mehta, A., y Dayal, U. (2008). Pqr: Predicting query execution times for autonomous workload management. En *2008 International Conference on Autonomic Computing*, pp. 13–22. IEEE.

[Hauke y Kossowski, 2011] Hauke, J. y Kossowski, T. (2011). Comparison of values of pearson's and spearman's correlation coefficients on the same sets of data. *Quaestiones geographicae*, 30(2):87–93.

[Hoffman *et al.*, 2013] Hoffman, M. D., Blei, D. M., Wang, C., y Paisley, J. (2013). Stochastic variational inference. *Journal of Machine Learning Research*.

[Janiesch *et al.*, 2021] Janiesch, C., Zschech, P., y Heinrich, K. (2021). Machine learning and deep learning. *Electronic Markets*, 31(3):685–695.

[Katzir *et al.*, 2020] Katzir, L., Elidan, G., y El-Yaniv, R. (2020). Net-dnf: Effective deep modeling of tabular data. En *International conference on learning representations*.

[Kingma y Welling, 2013] Kingma, D. P. y Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

[Lillicrap *et al.*, 2020] Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J., y Hinton, G. (2020). Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346.

[Lundberg y Lee, 2017] Lundberg, S. M. y Lee, S.-I. (2017). A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30.

[Matsunaga y Fortes, 2010] Matsunaga, A. y Fortes, J. A. (2010). On the use of machine learning to predict the time and resources consumed by applications. En *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pp. 495–504. IEEE.

[Mikolov *et al.*, 2013] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., y Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26.

[Morales y Zaragoza, 2012] Morales, E. F. y Zaragoza, J. H. (2012). An introduction to reinforcement learning. En *Decision Theory Models for Applications in Artificial Intelligence: Concepts and Solutions*, pp. 63–80. IGI Global.

[Nguyen *et al.*, 2018] Nguyen, T., Tran, N., Nguyen, B. M., y Nguyen, G. (2018). A resource usage prediction system using functional-link and genetic algorithm neural network for multivariate cloud metrics. En *2018 IEEE 11th conference on service-oriented computing and applications (SOCA)*, pp. 49–56. IEEE.

[Paisley *et al.*, 2012] Paisley, J., Blei, D., y Jordan, M. (2012). Variational bayesian inference with stochastic search. *arXiv preprint arXiv:1206.6430*.

[Pao, 1989] Pao, Y. (1989). Adaptive pattern recognition and neural networks.

[Pezoa *et al.*, 2022] Pezoa, R., Bórquez, S., Brooks, W., Salinas, L., y Torres, C. (2022). Uncertainty estimation in deep learning based-classifiers of high energy physics events using monte carlo dropout.

[Pisinger, 2005] Pisinger, D. (2005). Where are the hard knapsack problems? *Computers & Operations Research*, 32(9):2271–2284.

[Popov *et al.*, 2019] Popov, S., Morozov, S., y Babenko, A. (2019). Neural oblivious decision ensembles for deep learning on tabular data. *arXiv preprint arXiv:1909.06312*.

[Reinelt, 1995] Reinelt, G. (1995). Tsplib95. *Interdisziplinäres Zentrum für Wissenschaftliches Rechnen (IWR), Heidelberg*, 338:1–16.

[Rezende *et al.*, 2014] Rezende, D. J., Mohamed, S., y Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. En *International conference on machine learning*, pp. 1278–1286. PMLR.

[Rivin *et al.*, 1994] Rivin, I., Vardi, I., y Zimmermann, P. (1994). The n-queens problem. *The American Mathematical Monthly*, 101(7):629–639.

[Schmidt *et al.*, 2018] Schmidt, F., Niepert, M., y Huici, F. (2018). Representation learning for resource usage prediction. *arXiv preprint arXiv:1802.00673*.

[Sedgwick, 2012] Sedgwick, P. (2012). Pearson's correlation coefficient. *Bmj*, 345.

[Shapley, 1951] Shapley, L. S. (1951). Notes on the n-person game—ii: The value of an n-person game.

[Sharma *et al.*, 2017] Sharma, S., Sharma, S., y Athaiya, A. (2017). Activation functions in neural networks. *Towards Data Sci*, 6(12):310–316.

[Shwartz-Ziv y Armon, 2022] Shwartz-Ziv, R. y Armon, A. (2022). Tabular data: Deep learning is not all you need. *Information Fusion*, 81:84–90.

[Srivastava *et al.*, 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., y Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

[Titsias y Lázaro-Gredilla, 2014] Titsias, M. y Lázaro-Gredilla, M. (2014). Doubly stochastic variational bayes for non-conjugate inference. En *International conference on machine learning*, pp. 1971–1979. PMLR.