

**UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA**

CARRERA DE Ciencia de la Computación




**Modelos para evaluación de escalabilidad y  
precisión de Aproximaciones N-Body en GPU**


**TRABAJO DE INVESTIGACIÓN**

Para optar el grado de *Bachiller en Ciencia de la Computación*

**AUTOR**

Mauricio Bernardo Bernuy Geiser 

**ASESOR**

Jose Fiestas 

Lima - Perú

2024

*Dedicatoria:*

*some special people*

*Agradecimientos:*

*A mi familia, por darme apoyo durante todo este proceso Universitario.*

*A Mauricio, Ruben y Jordan por su apoyo durante el proceso de experimentación.*

# TABLA DE CONTENIDO

	Pág.
<b>RESUMEN</b> . . . . .	1
<b>ABSTRACT</b> . . . . .	2
<b>INTRODUCCIÓN</b> . . . . .	3
Presentación del tema de investigación . . . . .	3
Descripción de la situación problemática . . . . .	3
Formulación del problema . . . . .	4
Objetivos de investigación . . . . .	5
Alcance y limitaciones / restricciones . . . . .	6
 <b>CAPÍTULO I REVISIÓN CRÍTICA DE LA LITERATURA</b>	 8
1.0.1 Alcance de Estudios Previos . . . . .	12
1.0.2 Contribución de la tesis . . . . .	12
 <b>CAPÍTULO II MARCO TEÓRICO</b>	 14
2.1 Gravitational N-Body . . . . .	14
2.2 Barnes-Hut Approximation . . . . .	16
2.3 Quad-Tree / Oct-Tree . . . . .	17
2.4 Distribuciones de partículas . . . . .	19
2.4.1 Distribución Plummer . . . . .	19
2.5 Indicadores de precisión . . . . .	20
2.5.1 Energy Error . . . . .	20

2.6	Indicadores de ejecución . . . . .	20
2.6.1	Análisis de escalabilidad . . . . .	20
2.6.2	Perfilado en GPU . . . . .	21
2.7	Indicadores de Ejecución . . . . .	21
2.7.1	Características de GPU . . . . .	21
2.7.2	Parámetros de Simulación . . . . .	22
2.8	Técnicas de extracción de características . . . . .	22
2.8.1	Coefficiente de Correlación de Spearman . . . . .	22
2.8.2	Hierarchical Clustering . . . . .	22
2.8.3	Mutual Information Regression . . . . .	23
2.9	Técnicas de Regresión . . . . .	23
2.9.1	Regresión Lineal . . . . .	23
2.9.2	Regresión Polinomial . . . . .	23
2.9.3	Regresión con Máquinas de Soporte Vectorial . . . . .	24
2.9.4	Árboles de Decisión . . . . .	24
2.10	Técnicas de medición de errores para modelos . . . . .	24
2.11	Técnicas de Cross Validation . . . . .	25
2.11.1	Random Cross Validation . . . . .	25
2.11.2	Group K-Fold (Leave-One-Out) . . . . .	25
2.11.3	Timeseries Cross Validation . . . . .	26
2.12	Técnicas de Búsqueda de Hiperparametros . . . . .	27
2.12.1	Búsqueda Exhaustiva . . . . .	27
2.13	Diseño de Tesis . . . . .	27

## **CAPÍTULO III MARCO METODOLÓGICO 30**

3.1	Recopilación, Análisis y Procesamiento . . . . .	30
3.1.1	<i>Testbed</i> de ejecución . . . . .	31
3.1.2	Captura de características de hardware . . . . .	31
3.1.3	<i>Plan</i> de ejecución y captura de parámetros de ejecución . . . . .	32

3.1.4 Selección de Kernels a perfilar . . . . .	34
3.1.5 Captura de métricas de perfilado . . . . .	35
3.1.6 Construcción de <i>Dataset</i> completo . . . . .	36
3.1.7 Análisis y Preprocesamiento de características para ejecución . . . . .	37
3.1.7.1 Características de Simulación . . . . .	37
3.1.7.2 Características de GPU . . . . .	37
3.1.7.3 Características de Perfilado . . . . .	38
3.1.8 Análisis y Preprocesamiento de características para precisión . . . . .	39
3.1.8.1 Características de Simulación . . . . .	39
3.1.9 Características finales . . . . .	39
3.2 Modelado de ejecución y precisión . . . . .	40
3.2.1 Distribución de características . . . . .	40
3.2.2 Transformación y escalado de características . . . . .	41
3.2.3 Entrenamiento mediante Cross-Validation . . . . .	43
3.2.4 Modelos y rangos de hiperparámetros . . . . .	43
3.2.5 Categorías de modelos a entrenar . . . . .	45
3.2.5.1 Ejecución . . . . .	45
3.2.5.2 Precisión . . . . .	46
3.2.6 Presentación y comparativa de resultados . . . . .	47
3.2.6.1 Ejecución . . . . .	47
3.2.6.2 Precisión . . . . .	47
3.3 Software y evaluación de proyección . . . . .	49
3.3.1 <i>Overview</i> de Implementación . . . . .	49
3.3.2 Carga y Selección de Modelos . . . . .	50
3.3.3 Selección de Variable Principal . . . . .	50
3.3.4 Parámetros de Variable Principal y Características de Entrada . . . . .	51
3.3.5 Presets de GPU . . . . .	52
3.3.6 Salidas Esperadas del Visualizador . . . . .	52

3.3.6.1	Gráfica de Estimación . . . . .	52
3.3.6.2	Gráfica del Dataset . . . . .	53
3.3.7	Predicciones sobre GPUs Evaluadas . . . . .	53
3.3.8	Presets Adicionales de GPU . . . . .	54
3.3.9	Predicciones sobre GPUs No Evaluadas . . . . .	55
<b>CAPÍTULO IV RESULTADOS</b>		<b>56</b>
4.1	U1. Recopilación, Análisis y Perfilado . . . . .	56
4.1.1	Selección de Kernels . . . . .	56
4.1.2	Datasets finales . . . . .	57
4.1.2.1	Ejecución (2100 samples) . . . . .	57
4.1.2.2	Precisión (1000100 samples) . . . . .	57
4.1.3	Análisis y Preprocesamiento de Características de Ejecución . . . . .	58
4.1.3.1	N vs tiempo de simulación . . . . .	58
4.1.3.2	theta vs tiempo de simulación . . . . .	59
4.1.3.3	dt vs tiempo de simulación . . . . .	60
4.1.3.4	Observaciones Finales sobre características de simulación . . . . .	61
4.1.3.5	Características de GPU vs Tiempo de Simulación . . . . .	62
4.1.3.6	Características de perfilado vs tiempo de simulación . . . . .	64
4.1.4	Análisis y Preprocesamiento de Características de Precisión . . . . .	65
4.1.4.1	I vs Error Acumulado . . . . .	65
4.1.4.2	N vs Error Acumulado . . . . .	66
4.1.4.3	Theta vs Error Acumulado . . . . .	67
4.1.4.4	dt vs Error Acumulado . . . . .	68
4.1.4.5	Observaciones Finales sobre Características de Simulación . . . . .	69
4.2	U2. Entrenamiento de Modelos . . . . .	69
4.2.1	Distribución de características . . . . .	69

4.2.2 Transformaciones por Aplicar . . . . .	72
4.2.3 Distribución Transformada de Características . . . . .	72
4.2.4 Modelado de ejecución . . . . .	74
4.2.4.1 Ejecución sin Perfilado . . . . .	75
Modelo Lineal . . . . .	75
Modelo Polinomial . . . . .	75
Support Vector Machines (SVM) . . . . .	75
Árboles de Decisión . . . . .	76
Modelo Lineal . . . . .	76
Modelo Polinomial . . . . .	76
Support Vector Machines (SVM) . . . . .	77
Árboles de Decisión . . . . .	77
Extra Trees . . . . .	77
4.2.4.2 Ejecución con perfilado . . . . .	77
Modelo Lineal . . . . .	78
Modelo Polinomial . . . . .	78
Support Vector Machines (SVM) . . . . .	78
Árboles de Decisión . . . . .	78
Extra Trees . . . . .	78
Modelo Lineal . . . . .	79
Modelo Polinomial . . . . .	79
Support Vector Machines (SVM) . . . . .	79
Árboles de Decisión . . . . .	79
Extra Trees . . . . .	80
4.2.4.3 Comparación y Selección de Modelos de Ejecución . . . . .	80
4.2.5 Modelado de precisión . . . . .	81
Modelo Lineal: . . . . .	81
Modelo Polinomial: . . . . .	81



Support Vector Machines (SVM): . . . . .	82
Árbol de Decisión (Directo): . . . . .	82
Extra Trees: . . . . .	82
4.2.5.1 Comparación y Selección de Modelos de Precisión . .	83
4.3 U3. Visualización y Estimación . . . . .	84
4.3.1 Software de Visualización . . . . .	84
4.3.2 Visualización y Análisis de Modelos . . . . .	85
4.3.2.1 Ejecución . . . . .	85
4.3.2.2 Precisión . . . . .	86
4.3.3 Evaluación de Comportamiento de GPUs Fuera de Alcance . . . . .	88
4.3.3.1 Tesla T4 . . . . .	89
4.3.3.2 Tesla P100 . . . . .	90
4.3.3.3 A100 . . . . .	91
4.3.3.4 Evaluación de Comportamiento de <i>Selene</i> . . . . .	92
<b>CONCLUSIONES</b> . . . . .	94
<b>RECOMENDACIONES</b> . . . . .	97
<b>ANEXOS</b> . . . . .	103

## ÍNDICE DE TABLAS

Tabla 1.1	Resultados por Trabajo . . . . .	12
Tabla 3.1	GPUs de Entrenamiento . . . . .	31
Tabla 3.2	Rango de variables de ejecución . . . . .	33
Tabla 3.3	Rango de variables de precisión . . . . .	34
Tabla 3.4	Estructura de dataset de ejecución . . . . .	36
Tabla 3.5	Estructura de dataset de precisión . . . . .	37
Tabla 3.6	Transformaciones a disposición . . . . .	41
Tabla 3.7	Ejemplo de resultado principal . . . . .	48
Tabla 3.8	Detalle de resultado auxiliar . . . . .	48
Tabla 3.9	GPUs fuera de alcance . . . . .	54
Tabla 4.1	Top 5 kernels . . . . .	56
Tabla 4.2	Características completas de dataset de ejecución . . . . .	57
Tabla 4.3	Características completas de dataset de ejecución . . . . .	57
Tabla 4.4	Características de GPU mediante mutual_info_regression . . . . .	63
Tabla 4.5	Características finales de GPU . . . . .	63
Tabla 4.6	Características finales de Perfilado mediante correlación y clustering . . . . .	64
Tabla 4.7	Transformaciones por característica . . . . .	72
Tabla 4.8	Errores finales, utilizando Random CV y GPU CV . . . . .	75
Tabla 4.9	GPU CV, error por GPU . . . . .	76
Tabla 4.10	Errores finales con perfilado, utilizando Random CV y GPU CV . . . . .	77

Tabla 4.11 GPU CV, error por GPU . . . . .	79
Tabla 4.12 Errores finales, utilizando Random CV y TimeSeries CV . . . .	81
Tabla 4.13 TS CV, error por rango de iteraciones . . . . .	83

# ÍNDICE DE FIGURAS

Figura 2.1	Obtenido de [2]: Interacción gravitacional entre dos clusters de 2000 partículas. . . . .	15
Figura 2.2	Caso 1, el criterio evalúa $0.25/0.85 < 0.4$ como verdadero, tomando su centro de masa para la aproximación. Caso 2, se evalúa $0.25/0.5 < 0.4$ como falso, usando la partícula para el cálculo. . . . .	17
Figura 2.3	Ejemplo de cuadrantes en Quad-Tree . . . . .	18
Figura 2.4	Ejemplo de cuadrantes en Oct-Tree. . . . .	18
Figura 2.5	random CV, K=4 . . . . .	25
Figura 2.6	group CV, 4 Grupos . . . . .	26
Figura 2.7	Timeseries CV, 4 series . . . . .	26
Figura 2.8	Marco de Diseño Teórico . . . . .	27
Figura 3.1	<b>Primera fase:</b> Recopilación de datos . . . . .	30
Figura 3.2	gpu_info.json . . . . .	32
Figura 3.3	runs.json . . . . .	33
Figura 3.4	error.json . . . . .	33
Figura 3.5	Ejemplo de perfilado para Nsight compute . . . . .	35
Figura 3.6	<b>Segunda fase:</b> Análisis y preprocesamiento . . . . .	36
Figura 3.7	Esquema de selección mediante <i>mutual_info_regression</i> . . . . .	38
Figura 3.8	Esquema de selección de características de perfilado . . . . .	38
Figura 3.9	<b>Primera fase:</b> Análisis de distribución y transformaciones . . . . .	40
Figura 3.10	Ejemplo de distribución de N y subsecuente transformación normalizada . . . . .	41

Figura 3.11 <b>Segunda fase:</b> Entrenamiento y selección de modelos . . . . .	42
Figura 3.12 Esquema detallado de entrenamiento . . . . .	43
Figura 3.13 <b>Huber</b> , hiperparámetros de entrenamiento . . . . .	44
Figura 3.14 <b>Polynomial Ridge</b> , hiperparámetros de entrenamiento . . . . .	44
Figura 3.15 <b>Support Vector</b> , hiperparámetros de entrenamiento . . . . .	44
Figura 3.16 <b>Decision Tree</b> , hiperparámetros de entrenamiento . . . . .	44
Figura 3.17 <b>Extra Tree</b> , hiperparámetros de entrenamiento . . . . .	45
Figura 3.18 <b>Primera fase:</b> Implementación del visualizador y salidas es- peradas . . . . .	49
Figura 3.19 Selección de modelo . . . . .	50
Figura 3.20 Selector de variable principal . . . . .	51
Figura 3.21 Selección de parámetros de simulación y GPU . . . . .	51
Figura 3.22 Selección de preset de GPU . . . . .	52
Figura 3.23 Ejemplo de gráfica estimada y existente para ejecución . . . . .	53
Figura 3.24 <b>Segunda fase:</b> Validación de GPUs fuera de alcance . . . . .	54
Figura 4.1 Curva de N vs tiempo de simulación, agrupado por GPU (es- cala log) . . . . .	58
Figura 4.2 Curva de theta vs tiempo de simulación, agrupado por GPU . . . . .	59
Figura 4.3 Curva de dt vs tiempo de simulación, agrupado por GPU (es- cala log) . . . . .	60
Figura 4.4 Características de GPU para N, theta y dt fijo . . . . .	62
Figura 4.5 Curva de Iteraciones vs error acumulado, agrupado por dt y theta (escala log) . . . . .	65
Figura 4.6 Curva de N vs error acumulado, agrupado por dt y theta (escala log) . . . . .	66
Figura 4.7 Curva de theta vs error acumulado, agrupado por dt y N (escala log) . . . . .	67

Figura 4.8	Curva de dt vs error acumulado, agrupado por theta y N (escala log) . . . . .	68
Figura 4.9	Características de ejecución sin transformar . . . . .	70
Figura 4.10	Características de precisión sin transformar . . . . .	71
Figura 4.11	Características de ejecución transformadas . . . . .	73
Figura 4.12	Características de precisión transformadas . . . . .	74
Figura 4.13	Implementación Final . . . . .	84
Figura 4.14	Predicción y extrapolación N . . . . .	85
Figura 4.15	Predicción y extrapolación theta . . . . .	85
Figura 4.16	Predicción y extrapolación dt . . . . .	86
Figura 4.17	Predicción y extrapolación I . . . . .	86
Figura 4.18	Predicción y extrapolación N . . . . .	87
Figura 4.19	Predicción y extrapolación theta . . . . .	87
Figura 4.20	Predicción y extrapolación dt . . . . .	87
Figura 4.21	Graficas generadas para gpus semejantes . . . . .	89
Figura 4.22	Graficas generadas para gpus semejantes . . . . .	90
Figura 4.23	Grafica de gpu semejante y estimaciones bajo dos modelos distintos . . . . .	91
Figura 4.24	Grafica de gpu semejante y estimaciones bajo dos modelos distintos . . . . .	92

# RESUMEN

Este trabajo investiga el comportamiento de simulaciones *N-Body* mediante el algoritmo *Barnes-Hut* en GPU, con el objetivo de establecer modelos predictivos para estimar los tiempos de ejecución y errores de simulación en función de diversas configuraciones de hardware y parámetros de ejecución, utilizando técnicas de machine learning. Se lleva a cabo un análisis exhaustivo y perfilado de ejecución en múltiples sistemas, recopilando especificaciones de hardware, métricas de perfilado y parámetros de simulación, que son filtrados y categorizados mediante *Feature Extraction*. Posteriormente, se entrenan y evalúan cinco tipos de modelos —*Linear Regression*, *Polynomial Regression*, *Support Vector Regression*, *Decision Tree Regression* y *Extremely Randomized Tree Regression*— basados en un subconjunto de datos preprocesado, transformado y utilizando diferentes técnicas de *Cross-Validation* para asegurar la robustez del modelo. Finalmente, se desarrolla un software que permite ingresar características de hardware y configuraciones de ejecución, generando proyecciones gráficas del rendimiento esperado, que se utilizan tanto para validar los modelos como para ofrecer una herramienta pública que evalúe el comportamiento de la implementación de *Barnes-Hut* entrenada. Esta investigación aspira a contribuir al desarrollo de simulaciones científicas más eficientes en GPU, ante la escasez de estudios sobre análisis de ejecución y su variación en base a los recursos computacionales a disposición, y se propone publicar abiertamente todo el material generado, promoviendo la iteración y mejora continua a partir de los resultados de la investigación.

**Palabras clave:**

N-Body; Barnes-Hut; CUDA; Machine Learning;

# ABSTRACT

## Models for scalability and precision evaluation of GPU-based N-Body Approximations

This work investigates the behavior of *N-Body* simulations using the *Barnes-Hut* algorithm on GPUs, aiming to establish predictive models to estimate execution times and simulation errors based on various hardware configurations and execution parameters, leveraging machine learning techniques. An exhaustive analysis and profiling of executions are conducted across multiple systems, collecting hardware specifications, profiling metrics, and simulation parameters, which are filtered and categorized through *Feature Extraction*. Subsequently, five types of models — *Linear Regression*, *Polynomial Regression*, *Support Vector Regression*, *Decision Tree Regression*, and *Extremely Randomized Tree Regression*— are trained and evaluated using a preprocessed and transformed subset of data, applying different *Cross-Validation* techniques to ensure model robustness. Finally, a software tool is developed to input hardware characteristics and execution configurations, generating graphical projections of the expected performance. This tool is used both to validate the models and to offer a public resource for evaluating the behavior of the trained *Barnes-Hut* implementation. This research aims to contribute to the development of more efficient scientific simulations on GPUs, addressing the lack of studies on execution analysis and its variation based on available computational resources. All generated materials are intended to be openly published, promoting continuous iteration and improvement based on the research results.

### Keywords:

N-Body; Barnes-Hut; CUDA; Machine Learning;



# INTRODUCCIÓN

## Presentación del tema de investigación

Los recientes avances en hardware de aceleradores gráficos [1] presentan un alto potencial para mejorar significativamente el rendimiento y la eficiencia en una amplia gama de aplicaciones computacionales, un ejemplo claro siendo la simulación científica; por ejemplo, simulando eventos físicos a gran escala. Un aplicación muy estudiada de esta clase de simulaciones es *Gravitational N-Body* [2], el cual computa la interacción de las fuerzas gravitatorias entre cada par de cuerpos en un sistema masivo, logrando modelar la evolución de la posición y velocidad de cada cuerpo en cada unidad de tiempo de la simulación, con una complejidad de  $N^2$ .

Para usos más prácticos, se suelen usar algoritmos de aproximación optimizados como *Barnes-Hut* [2], el cual divide el espacio jerárquicamente por medio de las estructuras *Oct-Tree*(3D) o *Quad-Tree*(2D), permitiendo un almacenamiento y acceso eficiente a los puntos insertados al reducir el espacio de búsqueda en una región de datos. Finalmente, los cálculos son realizados mediante un criterio de aceptación basado en distancia, ahorrando un numero considerable de comparaciones en cada unidad de tiempo y obteniendo una complejidad de  $N\log N$ .

## Descripción de la situación problemática

Es importante destacar que proyectar el comportamiento de esta clase de algoritmos, especialmente en implementaciones sobre GPUs, sigue siendo un desafío complejo y poco explorado. Factores como la carga de trabajo y las particularidades del hardware impactan de manera significativa en el rendimiento, lo que dificulta predecir su eficiencia con precisión. En el caso específico del algoritmo *Barnes-Hut*

en GPU, las implementaciones más avanzadas utilizan *kernels* de alta complejidad para distribuir la carga de trabajo en partes del código que presentan limitaciones en su paralelización. Esto aprovecha al máximo la arquitectura del GPU, pero a la vez introduce una alta variabilidad en el comportamiento del algoritmo, siendo esta dependiente de las características específicas del GPU empleado, como el número de núcleos, la memoria y el ancho de banda.

### Formulación del problema

Uno de los principales retos al trabajar con algoritmos implementados en GPUs es la dificultad de proyectar el comportamiento del sistema debido a las diferencias en las características del hardware y los parámetros de ejecución de la simulación. Estas variaciones no solo complican la estimación precisa de los recursos necesarios para ejecutar una simulación, sino que también hacen casi imposible predecir el comportamiento de ejecución en simulaciones de mayor tamaño que exceden las capacidades del hardware actual. La naturaleza paralela del hardware GPU y la complejidad de la simulación agregan una capa adicional de incertidumbre en la previsión del rendimiento para configuraciones futuras.

Amaris [3] destaca un reciente enfoque emergente que utiliza modelos de *machine learning* para proyectar el comportamiento de *kernels* de GPU basándose en las características del hardware, permitiendo así estimar el tiempo de ejecución del código de manera más precisa. Siguiendo esta línea, vemos una oportunidad de expandir este enfoque al añadir parámetros específicos de ejecución del algoritmo *Barnes-Hut*, lo que nos permitiría modelar de forma más completa el comportamiento de la simulación para diferentes configuraciones de hardware y parámetros de ejecución. Esta integración podría facilitar una mejor planificación de recursos y una previsión más exacta del rendimiento en entornos complejos.

## Objetivos de investigación

El objetivo principal de esta investigación es modelar el comportamiento de una simulación N-Body basada en el algoritmo *Barnes-Hut* mediante el uso de técnicas de *machine learning*, con el fin de predecir los tiempos de ejecución para diversas configuraciones de hardware y parámetros de simulación. Para cumplir con este objetivo general, se han definido tres objetivos específicos, los cuales se desarrollan a continuación:

**Análisis de características de perfilado de ejecución en múltiples configuraciones de hardware:** El primer objetivo consiste en realizar un análisis detallado de los tiempos de ejecución y error de simulación en función del número de partículas ( $N$ ) y el parámetro *theta* (hiperparámetro de aproximación) en diversos tipos de hardware. Se llevará a cabo un perfilado exhaustivo de los recursos utilizados y de las características del equipo mediante herramientas de análisis y programas desarrollados en *Python*. En base a este conjunto de datos, se realizará un proceso de *Feature Extraction*, mediante el cual se establecerá un conjunto reducido de características representativas del perfilado. Este paso proporcionará una base de datos sólida que refleje cómo diferentes configuraciones de hardware influyen en el rendimiento del algoritmo y que características resultan más representativas para el posterior entrenamiento de nuestros modelos.

**Modelado de ejecución y precisión utilizando *Machine Learning*:** El segundo objetivo se enfoca en la creación de modelos predictivos utilizando *machine learning*. Basándonos en los datos obtenidos del perfilado, se entrenarán y evaluarán modelos de regresión *Lineales*, *Polinomiales*, *Support Vector Machines* (SVM) y *Decision Trees*, siguiendo un paradigma de *cross-validation* durante el entrenamiento para obtener modelos generalizables entre diferentes configuraciones de hardware. Estos modelos serán utilizados para predecir los tiempos de ejecución y los errores

de aproximación de la simulación *Barnes-Hut* en configuraciones de hardware y parámetros de simulación no explorados previamente. Se buscará comparar el error de predicción MAPE de dichos modelos para seleccionar el más adecuado para cada tarea.

**Desarrollo de software de proyección de ejecución:** El último objetivo implica la creación de un software que, utilizando el modelo predictivo final, permita ingresar las características del hardware y los parámetros de ejecución para obtener proyecciones gráficas de los tiempos de ejecución y precisión. Este software estará diseñado para ser fácil de usar y flexible, facilitando así la evaluación del rendimiento para futuras simulaciones sin necesidad de realizar pruebas en cada configuración posible de hardware.

Estos tres objetivos, aunque independientes, se complementan para proporcionar una solución integral al problema planteado, abarcando desde la recolección de datos hasta la implementación de un software funcional que proyecte el comportamiento del algoritmo en distintas condiciones.

Se considera que este trabajo aportará valor al fomentar evaluaciones más precisas y efectivas en esta área de investigación, que cuenta actualmente con pocos estudios enfocados en el análisis de ejecución y uso de recursos en simulaciones de este tipo. Además, buscamos que todo el material, tanto los datos como el software desarrollado, sea publicado de forma abierta para facilitar futuras iteraciones y avances en este campo.

### **Alcance y limitaciones / restricciones**

En base a las limitaciones y el alcance de la presente propuesta, trabajaremos bajo las siguientes restricciones:

- Durante la experimentación, se evaluará la simulación bajo distribuciones de partículas replicables, siguiendo la distribución de *Plummer Spheres* [4, 5] para mantener resultados comparables y fáciles de replicar.
- Se enfocará la experimentación en arquitecturas *single GPU*, utilizando GPUs de gama comercial como entornos de prueba.
- Se buscará desarrollar una experimentación fácil de analizar y replicar, publicando el código y resultados de manera abierta para facilitar posteriores trabajos en el área.

# CAPÍTULO I

## REVISIÓN CRÍTICA DE LA LITERATURA

Este capítulo busca presentar los trabajos relacionados en el área para evaluar el estado actual de las simulaciones *N-Body* en GPU y la evaluación de kernels de GPU, para poder plantear de manera correcta la posterior experimentación y evaluación de resultados.

Haciendo una evaluación de varias implementaciones conocidas de la simulación N-Body, la fase de experimentación de la aproximación Barnes-Hut en GPU **Bonsai** [5], considerada por múltiples estudios como una implementación altamente optimizada y apta para comparaciones de desempeño [4, 6], hace énfasis en la medición de ejecución por etapas del algoritmo, planteando cinco distintas fases de su implementación: *Sorting*, *Moving*, *Construction* (*fase de creación de estructura*), *Properties* y *Tree Transverse*. Este también plantea una evaluación del Escalado de tiempos de ejecución vs el tamaño de la simulación de manera separada para estas fases identificadas del algoritmo. Múltiples otras experimentaciones, como las presentadas en: [2], [6] (Gflops, métrica equivalente), utilizan esta métrica de Escalado para demostrar el desempeño de sus implementaciones, pero solo toman la métrica de tiempo total. Recalcando la propuesta de [5], analizar el tiempo dividido en fases permite analizar mejoras en etapas específicas del algoritmo, como por ejemplo una mejora de *trasversal* a costo de mayor tiempo de *construction*. Olsson [4] apoya esto al encontrar una relación entre el costo de construcción de aproximaciones Barnes-Hut y el cálculo de interacciones, afectando los tiempos de ejecución totales de cada iteración de la simulación, identificando de esta manera una relación entre diferentes fases de la simulación y los tiempos totales de ejecución.

Al desarrollar un método aproximado para una simulación real, resulta importante poder validar la precisión obtenida, con el objetivo de validar su viabilidad de uso en una simulación que guarde coherencia con el método directo. Esto resulta un área de evaluación importante en cada propuesta de implementación de simulaciones N-Body, en especial las aproximaciones como Barnes-Hut. Bonsai [5] plantea la métrica de **Acceleration Error**, para poder comprobar la calidad de la aproximación del algoritmo implementado frente al cálculo directo de fuerzas, al finalizar un número determinado de iteraciones. Esta métrica recomienda usar un algoritmo de Direct Computation de doble precisión como **Sapporo** [7] en CPU para obtener las aceleraciones finales de calculo directo. De manera similar, [5] y otra serie de trabajos [2, 8], utilizan la métrica de **Energy Error**, la cual permite medir la pérdida de energía por aproximación del algoritmo a través de sus iteraciones. Un cálculo directo de N-Body debería presentar una pérdida de energía prácticamente nula, por lo cual esta puede utilizarse como una métrica similar al error de aceleración, pero también puede ser usada para identificar variaciones considerables de energía durante ciertas etapas de la simulación [5].

A su vez, la experimentación de simulaciones y aproximaciones N-Body suelen ser evaluadas mediante el uso de distribuciones conocidas y replicables de partículas, tales como *Uniform Distribution*, *Plummer Distribution* y *Galaxy Merger*. De estas, la más generalizada para un contexto astronómico resulta en la distribución Plummer de modelos globurales de estrellas [9], utilizada como la distribución de partículas mas generalizada en diferentes propuestas de implementación por su amplio estudio y facilidad de replicación [2, 4–7, 10]

En el contexto de estimaciones de ejecución de programas en GPU, se destaca un reciente enfoque emergente en utilizar modelos de *machine learning* para proyectar el comportamiento de estos *kernels*. Estos estudios difieren en el enfoque dado, variando entre buscar la generalización de diferentes kernels con un enfoque

en portabilidad entre arquitecturas [11], favoreciendo un análisis estático de ejecución para GPUs desconocidas [12], y diferenciando la estimación de GPUs y Kernels desconocidos bajo diferentes modelos en base a *Feature Extraction* [3],

El artículo de Amaris et al. [3] compara la precisión de un modelo analítico y tres técnicas de *machine learning* (regresión lineal, máquinas de soporte vectorial y bosque aleatorio) para predecir los tiempos de ejecución de *kernels* en GPUs. La evaluación se realizó en un conjunto de 20 *kernels* ejecutados en nueve GPUs distintas. Para abordar el problema de generalización a hardware desconocido o a *kernels* no previamente evaluados, los autores entrenaron modelos utilizando una variante de validación cruzada *leave-one-out*. En este método, una GPU o un *kernel* completo se excluye durante el entrenamiento y se utiliza exclusivamente en el conjunto de evaluación, permitiendo probar la capacidad de generalización de los modelos a configuraciones no vistas. El estudio emplea extracción de características para identificar las variables de perfilado y de hardware más relevantes, minimizando el riesgo de sobreajuste. Los resultados muestran que el modelo de regresión lineal, optimizado con la selección de dos parámetros clave de hardware (caché L2 y número de núcleos) junto con entre cinco y diez variables de perfilado, alcanzó la mayor precisión en el caso de GPUs desconocidas, con un error MAPE de entre 1.37 % y 1.65 %. Este enfoque evidencia la efectividad de técnicas de reducción de características en modelos de predicción de rendimiento, maximizando la precisión y robustez del modelo en el contexto de arquitecturas y *kernels* no evaluados.

Por otro lado, el trabajo de Braun et al. [11] presenta un modelo de predicción portátil y eficiente para estimar el tiempo de ejecución y el consumo energético de *kernels* en GPUs, independiente de la arquitectura del hardware. Su enfoque se basa en la regresión de árboles extremadamente aleatorios (*Extremely Randomized Trees Regression*), utilizando métricas de perfilado del conjunto de instrucciones *PTX* para asegurar la portabilidad entre diferentes plataformas de GPU. El modelo está



configurado con parámetros optimizados para maximizar su precisión y generalización, como el número de estimadores (128, 256, 512 o 1,024), criterios de división ( $MSE$  o  $MAE$ ) y el número máximo de características consideradas ( $max$ ,  $\log 2$  o  $\sqrt{}$ ). Sin embargo, no se realiza una selección de características exhaustiva ni se eliminan variables irrelevantes en la entrada del modelo, aunque se identifican las variables más influyentes en una evaluación posterior. Además, se emplea una función de pérdida  $L1$  para manejar los valores atípicos y se evalúa el modelo mediante el error porcentual medio absoluto (MAPE) como métrica principal. La evaluación del modelo se realizó en cinco arquitecturas de GPU y cuatro conjuntos de *kernels*, obteniendo un rango de error MAPE entre 8.86 % y 52 % en la predicción del tiempo de ejecución. Este rango de error se atribuye principalmente a limitaciones en la capacidad del modelo para capturar propiedades específicas de las GPUs y a un desequilibrio en las clases de *kernels* utilizados.

A su vez, Lanker et al. [12] presentan un modelo para proyectar el rendimiento de *kernels* de GPU en arquitecturas de hardware distintas, evaluando su comportamiento en una GPU base y extendiéndolo a una GPU destino. La propuesta se basa en un modelo jerárquico que considera tanto el balance de carga computacional como la intensidad de operaciones de memoria. Para esto, utilizan variables de perfilado y un análisis estático del código de compilación del *kernel*. Este modelo jerárquico se estructura en dos etapas: primero, se evalúan las métricas de perfilado en la GPU base, permitiendo modelar el *kernel* en el contexto del hardware inicial. Luego, se realiza una segunda evaluación en la que el código compilado se analiza tanto en la GPU base como en la GPU destino, combinando ambos modelos para proyectar el comportamiento esperado en la GPU objetivo. La evaluación del enfoque incluyó cuatro arquitecturas de GPU y cinco conjuntos de *kernels*, obteniendo un rango de error MAPE de 5.9 % a 20.3 % en la predicción de rendimiento. A pesar de estos resultados prometedores, los autores sugieren que ciertos errores de proyección pueden estar relacionados con la ausencia de métricas adicionales en el modelo, como

la ocupación de GPU y el comportamiento de memoria y caché. Estas omisiones limitan la capacidad del modelo para capturar de manera más precisa las diferencias en la utilización de los recursos entre GPUs.

### 1.0.1 Alcance de Estudios Previos

Para resumir el alcance y resultados de estudios previos mas relevantes, se presenta la siguiente tabla:

	<b>Amaris et al.</b>	<b>Braun et al.</b>	<b>Lanker et al.</b>
<b>Models</b>	Linear, SVM, RF	ExtraTrees	Heirarchical Analysis
<b>MAPE</b>	1.37 % - 1.65	8.86 % - 52 %	5.9 % - 20.3 %
<b>Features</b>	Feature Extraction Profiling and GPU	Profiling and PTX Instructions	Profiling (source) and Compile result (source and target)
<b>Validation</b>	LOOV by GPU	Model Parameter Exhaustive Search	train test split
<b>Generalization</b>	Kernel and GPU	GPU	GPU

TABLA 1.1: Resultados por Trabajo

### 1.0.2 Contribución de la tesis

Con el objetivo de mejorar los resultados actuales, se busca implementar técnicas presentadas en los trabajos relacionados, adaptándolas al desarrollo de esta investigación. Esto incluye la utilización de modelos de machine learning, técnicas de extracción de características y procedimientos de validación, aplicados específicamente al contexto de la simulación Barnes-Hut en GPU.

La simulación Barnes-Hut involucra múltiples kernels ejecutados de manera secuencial, lo que la convierte en un caso de aplicación directa para las técnicas previamente mencionadas.

Los modelos finales desarrollados serán evaluados en función de los rangos de error MAPE reportados en los trabajos relacionados, lo que permitirá establecer una comparación cuantitativa y validar su eficacia en este contexto.

# CAPÍTULO II

## MARCO TEÓRICO

Presentamos a detalle los conceptos más importantes relevantes al presente trabajo.

### 2.1 Gravitational N-Body

El problema N-Body presenta un sistema en el que un número  $N$  de cuerpos o partículas con posición arbitraria, se encuentran interactuando mutuamente como un sistema dinámico, regido por alguna fuerza que gobierne dicho entorno.

Para un caso gravitacional[2], el objetivo sería calcular las fuerzas gravitacionales entre cada par de partículas en el sistema utilizando la Ley Gravitacional de Newton, su magnitud siendo dada como

$$F = \frac{Gm_i m_j}{r^2}$$

tomando dos partículas de masas  $m_i, m_j$  separadas por una distancia  $r$  ( $G$  = constante gravitacional). En base a esto, obtener la fuerza ejercida por la partícula  $j$  sobre  $i$  se daría multiplicando por su vector unitario de la siguiente manera

$$F_{ij} = \frac{Gm_i m_j}{||r_i - r_j||^2} \cdot \frac{(r_j - r_i)}{||r_i - r_j||}$$

siendo  $r_i, r_j$  las posiciones de cada respectiva partícula.

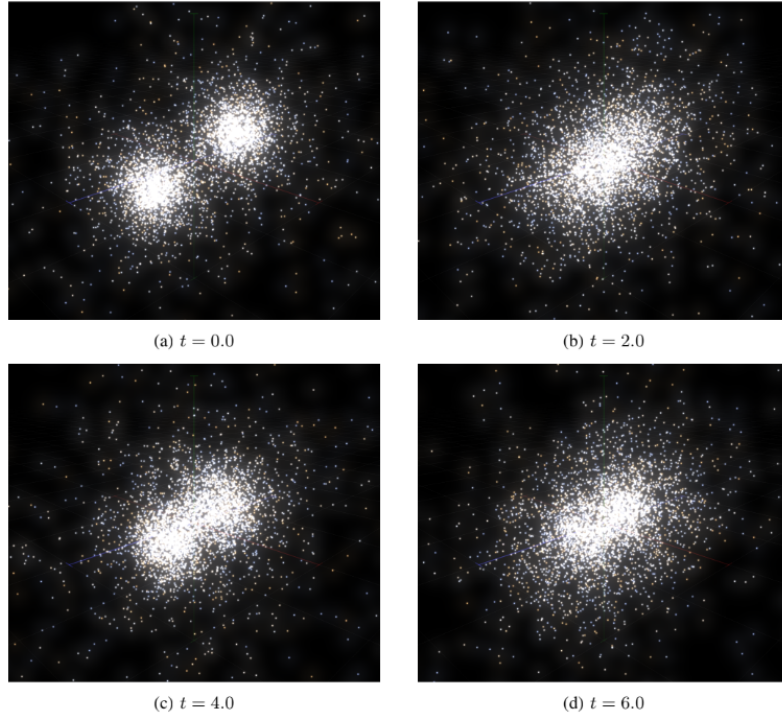


FIGURA 2.1: Obtenido de [2]: Interacción gravitacional entre dos clusters de 2000 partículas.

Despejando las ecuaciones diferenciales que rigen esta interacción, se logra establecer las siguientes ecuaciones para actualizar los valores de aceleración, posición y velocidad de una partícula de índice  $i$  para una unidad de tiempo en la simulación.

$$a_i = a_i + \sum_{j=1, j \neq i}^N \frac{Gm_j(r_j - r_i)}{||r_i - r_j||^3}$$

$$r_i = r_i + v_i + a_i/2$$

$$v_i = v_i + a_i$$

Esta actualización se calcula para cada partícula del sistema en una unidad de tiempo, de lo cual deriva su complejidad de  $N^2$ . Estos cálculos se repiten por cada paso de la duración establecida para la simulación, de esta manera calculando la evolución del sistema a través del tiempo.

## 2.2 Barnes-Hut Approximation

La complejidad de  $N^2$  presente en el algoritmo original resulta prohibitiva para simulaciones de escala mayor, debido a lo cual existen aproximaciones jerárquicas [2, 13] que buscan acelerar el cálculo de la simulación, agrupando grupos distantes de partículas como un solo centro de masa.

*Barnes-Hut*, de esta manera, particiona el espacio del sistema a través de estructuras de árboles jerárquicas, comúnmente mediante el uso de Quad-Trees para dos dimensiones y Oct-Trees para tres dimensiones [14]. Hablamos más adelante sobre estas estructuras a detalle. La distribución jerárquica de estas estructuras resulta en una representación de área o volumen de espacio cada vez mas grande dependiendo del nivel del árbol. En base a esta propiedad, la métrica de separación *multipole acceptance criterion (MAC)*[2] dada como

$$\frac{\ell}{d} < \theta$$

tomando  $\ell$  como la longitud lateral de un nodo,  $d$  como la distancia entre una partícula y el centro de masa de un nodo, y  $\theta$  como el parámetro *opening angle*, nos permite calcular directamente la interacción entre una partícula  $i$  y un nodo del árbol, interpretando dicho nodo como una sola partícula masiva si se cumple el criterio (nodos distantes), o recorriendo recursivamente sus nodos hijo hasta cumplir el criterio o se lleguen a los nodos hoja, en los cuales se calcula directamente la interacción partícula-partícula (nodos cercanos).

De esta manera, logramos apreciar como este algoritmo calcula a través de las mencionadas aproximaciones la interacción entre las partículas del sistema, excepto en casos donde estas se encuentren muy cercanas, momento en el cual las interacciones se calculan directamente entre partículas al igual que el algoritmo base. A

## Quad-Tree Barnes-Hut

$$\theta = 0.4$$

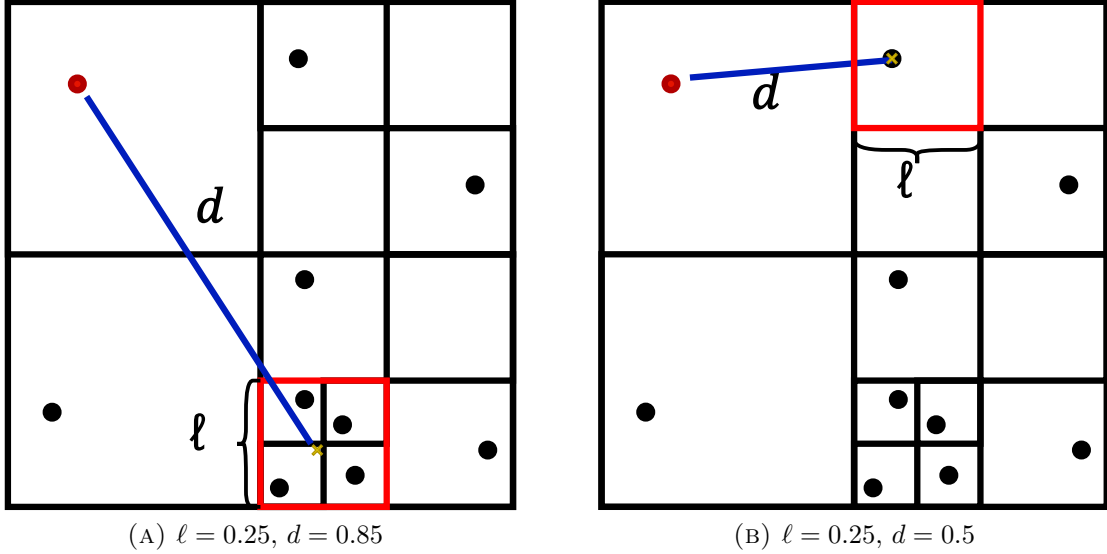


FIGURA 2.2: Caso 1, el criterio evalúa  $0.25/0.85 < 0.4$  como verdadero, tomando su centro de masa para la aproximación. Caso 2, se evalúa  $0.25/0.5 < 0.4$  como falso, usando la partícula para el cálculo.

partir de esto se deriva su reducción en complejidad a  $N \log N$ . Notar que, mientras más grande sea el valor  $\theta$  del criterio *MAC*, mayor sería el error de aproximación del algoritmo, pero a su vez reduciendo los cálculos necesarios para la simulación.

### 2.3 Quad-Tree / Oct-Tree

Los Quad-Tree y Oct-Tree son estructuras jerárquicas de tipo árbol, las cuales son usadas para particionar puntos, partículas u objetos distribuidos en un espacio de dos o tres dimensiones, respectivamente. Esto se logra, para un Quad-Tree, mediante una estructura de árbol particionada en cuatro cuadrantes, generando de manera recursiva hasta cuatro hijos por cada nodo del árbol [14]. De esta manera se logra hacer una subdivisión del espacio mediante la cual se genera una estructura en la cual cada celda contenga a lo mucho un solo objeto.

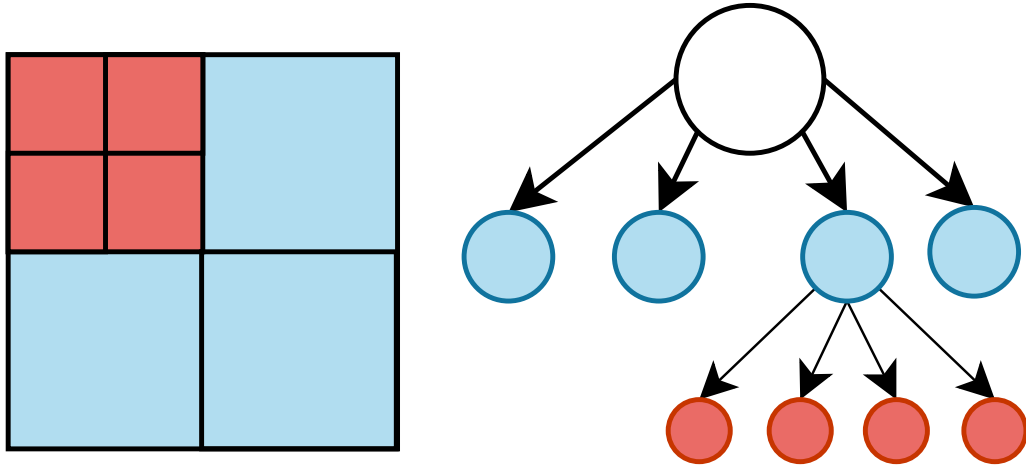


FIGURA 2.3: Ejemplo de cuadrantes en Quad-Tree

Para un Oct-Tree, el concepto resulta el mismo, pero particionando en hasta 8 cuadrantes por cada nodo del árbol.

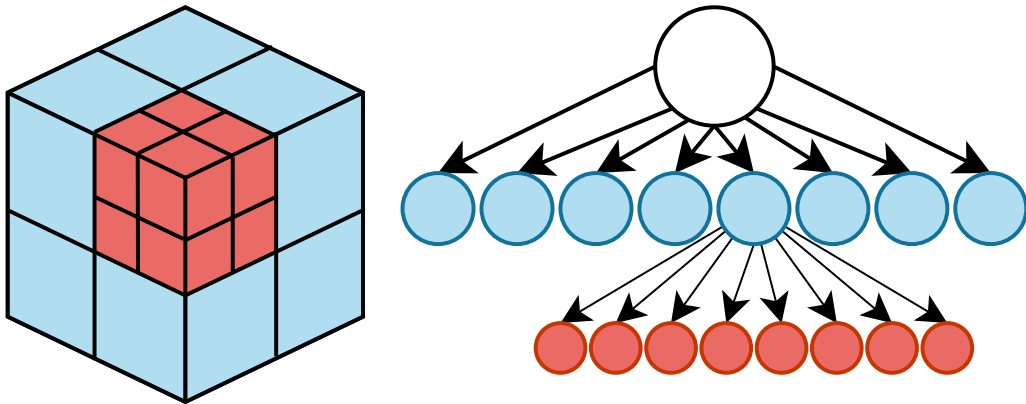


FIGURA 2.4: Ejemplo de cuadrantes en Oct-Tree.

La jerarquía presentada por estas estructuras permite un almacenamiento y acceso eficiente a los puntos insertados, pues este particionamiento permite reducir el espacio de búsqueda en una región de datos. Por otro lado, es importante denotar que estas estructuras no son muy eficientes en un contexto de GPU, en cuanto a la actualización o reconstrucción de sus puntos debido a su partición fija en cuadrantes, por lo cual ciertas optimizaciones son necesarias para su uso en un sistema dinámico, como sería el caso en una simulación. Estas optimizaciones varían de la



implementación y son fuertemente afectadas por diferencias en el hardware que las ejecutan.

## 2.4 Distribuciones de partículas

### 2.4.1 Distribución Plummer

El modelo de distribución Plummer es una función utilizada para describir la distribución de partículas en un sistema gravitacional, como un cúmulo estelar. Propone que la densidad de partículas  $\rho$  en un espacio tridimensional se distribuye de acuerdo con la siguiente relación:

$$\rho(r) = \frac{3M}{4\pi a^3} \left(1 + \frac{r^2}{a^2}\right)^{-5/2}$$

donde  $M$  es la masa total del sistema,  $a$  es un parámetro de escala que determina el tamaño del sistema, y  $r$  es la distancia radial desde el centro del cúmulo. Esta función muestra que la densidad de partículas es alta en el centro y disminuye suavemente hacia el exterior, lo que refleja la estructura de muchos sistemas astrofísicos.

La distribución Plummer fue introducida por el astrónomo inglés William Plummer en 1911 en su artículo titulado *On the Problem of Distribution of Stars in a Globular Cluster* [9]. Este modelo ha sido ampliamente utilizado en estudios astrofísicos para simular cúmulos estelares y otras estructuras galácticas, debido a su simplicidad y capacidad para representar de manera efectiva la distribución de masa en estos sistemas.

## 2.5 Indicadores de precisión

### 2.5.1 Energy Error

El *Energy Error* se define como la diferencia relativa entre la energía total esperada del sistema y la energía total calculada por el algoritmo de simulación al final de un intervalo de tiempo. La energía total del sistema en un contexto N-Body generalmente incluye la energía cinética y la energía potencial de todos los cuerpos en el sistema. Matemáticamente, el *Energy Error* se puede expresar como:

$$E_{\text{error}} = \frac{|E_{\text{total\_end}} - E_{\text{total\_start}}|}{E_{\text{total\_end}}}$$

tomando  $E_{\text{total\_start}}$  es la energía total inicial del sistema, calculada al comienzo de la simulación, y  $E_{\text{total\_end}}$  como la energía total calculada por el algoritmo al final del intervalo establecido de simulación.

## 2.6 Indicadores de ejecución

### 2.6.1 Analisis de escalabilidad

El análisis de escalabilidad en GPUs difiere notablemente del realizado en CPUs, debido a la naturaleza de las GPUs como sistemas de procesamiento masivamente paralelos. En este contexto, los lenguajes de programación diseñados para GPUs buscan abstraer al máximo el manejo del scheduling y el uso de núcleos, lo que dificulta la comparación directa con el rendimiento de algoritmos secuenciales. Por lo tanto, es fundamental evaluar métricas específicas como la ocupación de los multiprocesadores de streaming (SM) y realizar pruebas con diversas configuraciones de hardware para entender el comportamiento del código en entornos paralelos.

En consecuencia, no es factible establecer un tiempo de ejecución (runtime) serial para un código ejecutado en GPU, ya que la paralelización y la arquitectura del hardware introducen variables adicionales que afectan su rendimiento. Este enfoque requiere una comprensión más profunda de cómo las diferentes configuraciones de hardware y la gestión de recursos impactan en la ejecución de los algoritmos en este tipo de arquitecturas.

### 2.6.2 Perfilado en GPU

Con el objetivo de obtener un conjunto de métricas de ejecución, se hace uso de herramientas de perfilado de NVIDIA CUDA. Estas habilitan la grabación de múltiples métricas de hardware y ejecución de uno o múltiples Kernels de un programa designado y guardando estos datos para un posterior análisis. Entre estas herramientas se encuentra *nvprof* [15] y *NSight Compute* [16], diferenciadas debido a su uso por diferentes arquitecturas, necesitando utilizar una u otra dependiendo de la CUDA Capability de cada GPU es menor o mayor a 7.0.

## 2.7 Indicadores de Ejecución

### 2.7.1 Características de GPU

Las características de la GPU son obtenidas a través del API de CUDA[15], la cual permite acceder a diversas métricas relacionadas con el hardware. Entre las características relevantes se encuentran el tipo de arquitectura, la cantidad de memoria disponible, el número de núcleos de procesamiento, y el rendimiento en términos de operaciones por segundo. Estas características son cruciales definir las características y capacidades del hardware evaluado.

### 2.7.2 Parámetros de Simulación

Los parámetros de simulación [5] utilizados en este estudio incluyen:

- $N$ : Tamaño del problema, que determina el número de partículas en el sistema de simulación.
- $\theta$ : Parámetro de aproximación que ajusta el nivel de precisión en los cálculos.
- $\Delta t$  (dt): Tamaño del paso temporal, que controla la resolución temporal de la simulación.
- $I$ : Número de iteraciones, que representa la cantidad de ciclos de simulación realizados.

## 2.8 Técnicas de extracción de características

### 2.8.1 Coeficiente de Correlación de Spearman

Esta técnica mide la dependencia monotónica entre dos variables. A diferencia del coeficiente de correlación de Pearson, Spearman considera relaciones no lineales, siendo útil para identificar características con patrones de comportamiento que no necesariamente siguen distribuciones normales [3].

### 2.8.2 Hierarchical Clustering

Se utiliza para agrupar características similares en base a distancias métricas, permitiendo identificar grupos de variables con comportamiento redundante o altamente correlacionado. Este enfoque organiza las variables en una estructura jerárquica, lo que facilita la selección de una representación mínima del conjunto de datos.

### 2.8.3 Mutual Information Regression

Esta métrica mide la cantidad de información compartida entre una variable independiente y la variable objetivo, capturando tanto relaciones lineales como no lineales. Es útil para identificar características que aporten información única y relevante al modelo de predicción. [17]

## 2.9 Técnicas de Regresión

### 2.9.1 Regresión Lineal

La regresión lineal se puede aplicar utilizando, por ejemplo, una variante robusta conocida como *Huber Regression*[18], que combina un modelo lineal clásico L2 con un manejo adecuado de valores atípicos. Este método minimiza una función de pérdida insensible a grandes desviaciones, garantizando una mayor estabilidad en presencia de ruido.

### 2.9.2 Regresión Polinomial

La regresión polinomial emplea una combinación de transformación de características (*Polynomial Features*[18]) y regularización a través de un estimador lineal como *Ridge Regression*. Este enfoque permite capturar comportamientos no lineales en los datos mientras controla el sobreajuste al aplicar una penalización a los coeficientes del modelo.

### 2.9.3 Regresión con Máquinas de Soporte Vectorial

Se utiliza *Support Vector Regression* (SVR)[19] con un kernel de base radial (RBF), que permite modelar relaciones no lineales entre las variables independientes y la variable objetivo. Este método maximiza el margen entre las predicciones y los datos reales, proporcionando un balance entre complejidad y generalización.

### 2.9.4 Árboles de Decisión

Se evalúan modelos basados en *Decision Tree Regression*[20], los cuales generan predicciones discretas mediante una estructura jerárquica que divide los datos en subconjuntos homogéneos. Además, se emplean *Extremely Randomized Decision Trees*[21], una variante que introduce aleatoriedad adicional en el proceso de construcción del árbol, lo que mejora la robustez del modelo y su capacidad de generalización.

## 2.10 Técnicas de medición de errores para modelos

El **Error Absoluto Medio (MAE)**[22] mide la magnitud promedio de los errores en un conjunto de predicciones sin considerar su dirección. Se define como:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

donde  $y_i$  es el valor real,  $\hat{y}_i$  es el valor predicho, y  $n$  es el número de observaciones.

El **Error Porcentual Absoluto Medio (MAPE)**[22] mide el error relativo promedio en forma de porcentaje y se calcula como:

$$\text{MAPE} = \frac{1}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right| \times 100$$

Esta última métrica resulta útil para evaluar el rendimiento en diferentes escalas de datos, facilitando la interpretación del error obtenido.

## 2.11 Técnicas de Cross Validation

Se presentan las siguientes técnicas de *Cross Validation* [23]

### 2.11.1 Random Cross Validation

Este método selecciona aleatoriamente subconjuntos de entrenamiento y validación en cada iteración. Es adecuado para datos independientes y distribuidos uniformemente, permitiendo una evaluación robusta del modelo sobre diferentes particiones del conjunto de datos.

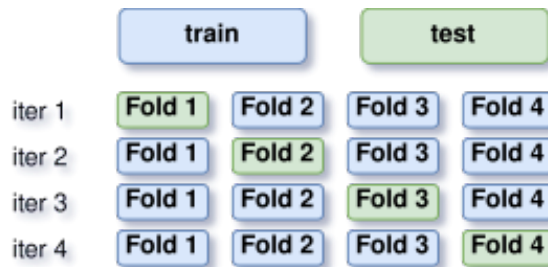


FIGURA 2.5: random CV, K=4

### 2.11.2 Group K-Fold (Leave-One-Out)

En este enfoque, los datos se agrupan en categorías o grupos específicos, y cada iteración deja un grupo fuera como conjunto de validación mientras entrena con

los restantes. Es ideal para escenarios donde los datos dentro de cada grupo están correlacionados, asegurando una evaluación más realista y sin fugas de información.

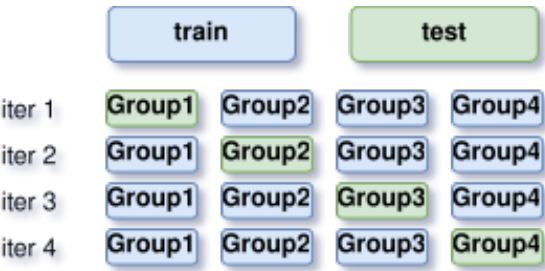


FIGURA 2.6: group CV, 4 Grupos

2.11.3 Timeseries Cross Validation

Diseñado para datos secuenciales, este método adapta la validación a una variable continua, garantizando que los conjuntos de entrenamiento siempre precedan temporalmente a los de validación. Es particularmente útil para problemas dependientes del tiempo, como series temporales o simulaciones iterativas, respetando la estructura cronológica de los datos.

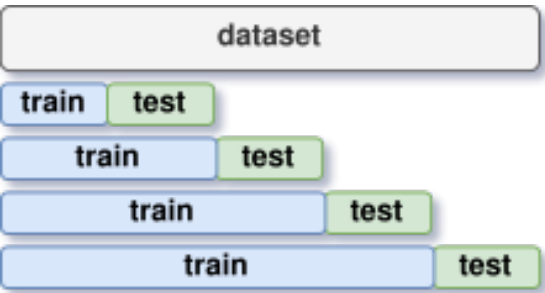


FIGURA 2.7: Timeseries CV, 4 series



## 2.12 Técnicas de Búsqueda de Hiperparámetros

### 2.12.1 Búsqueda Exhaustiva

La búsqueda exhaustiva [24] consiste en la evaluación sistemática de un rango listado de hiperparámetros. Para cada combinación de parámetros establecidos, se entrena el modelo y se seleccionan los hiperparámetros que minimizan el error. Esta técnica puede extenderse para evaluar combinaciones de características de entrada y calcular el error utilizando un método de *Cross Validation*.

## 2.13 Diseño de Tesis

Presentamos la siguiente gráfica de diseño de tesis, la cual diagrama la estructura planeada que seguiremos para cumplir nuestros objetivos de investigación.

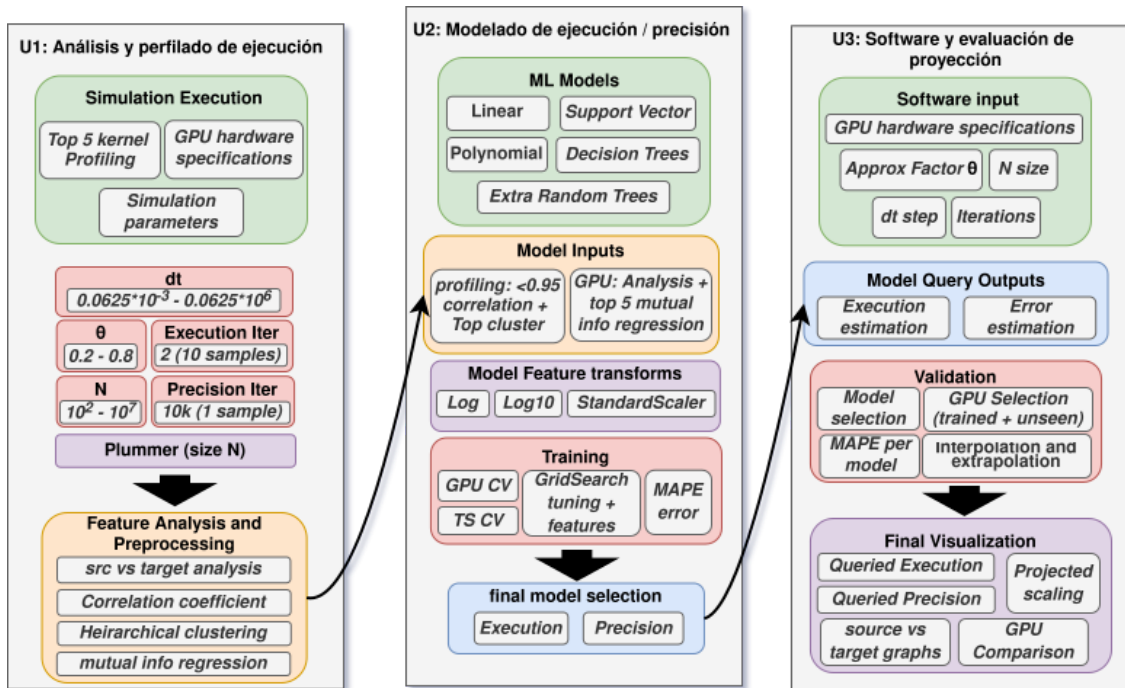


FIGURA 2.8: Marco de Diseño Teórico

## **U1: Recopilación, Análisis y Procesamiento de características de ejecución de Barnes-Hut**

Se realiza una recopilación exhaustiva del perfilado de los kernels más relevantes de la simulación, capturando tanto las características de hardware como los parámetros de simulación para un rango representativo de configuraciones de ejecución en cada GPU perfilada. Para garantizar la consistencia en los datos de entrada, se utilizan distribuciones de Plummer en la generación de la simulación, permitiendo una evaluación coherente del desempeño en todas las configuraciones de hardware. Las diferentes características encontradas serán sometida a un análisis de su comportamiento, seguido por una secuencia de preprocesamiento de datos en el que se aplica el coeficiente de correlación de Spearman para identificar y retener variables con relevancia significativa. Posteriormente, se realiza un proceso de *clustering* para agrupar parámetros similares y reducir la redundancia en los datos. Ciertos grupos de características más reducidas serán analizadas y filtradas mediante un análisis de *mutual info regression*. Esta base de datos limpia y estructurada constituye la fuente principal para el entrenamiento de los modelos predictivos.

## **U2: Entrenamiento y Optimización de Modelos Predictivos**

Se entrenarán dos clases modelos predictivos, uno enfocado en la estimación del tiempo de ejecución y otro en la predicción del error de simulación. El entrenamiento se lleva a cabo utilizando cuatro enfoques de modelado por regresión—*Lineal*, *Polinomial*, *Support Vector* y *Decision Tree*—sobre el conjunto de características preprocesadas. Las características de entrenamiento de estos modelos recibirán una secuencia de transformación y escalado a partir de un análisis previo de distribución, buscando normalizar la distribución de datos para facilitar el entrenamiento de los modelos. El proceso de entrenamiento se realiza mediante tres variaciones de *cross-validation*—*Random CV*, *Grouped CV* y *TimeSeries CV*—en conjunto de una búsqueda exhaustiva de hiperparámetros y subconjuntos de características

minimizando el error MAPE (Mean Absolute Percentage Error). Este *framework* de entrenamiento busca minimizar el *overfitting* de nuestros modelos y garantizar la generalización de sus predicciones. Al finalizar este proceso, se seleccionan modelos candidatos para su evaluación mediante el software desarrollado en el siguiente objetivo.

### **U3: Desarrollo de Herramientas para Validación y Visualización del Modelo**

Se implementa un software para facilitar tanto la validación como la visualización de los modelos predictivos, permitiendo consultas basadas en especificaciones de hardware y parámetros de ejecución, generando así valores estimados de ejecución y precisión. Este software permitirá visualizar los resultados predictivos de los modelos elegidos, permitiendo generar gráficos que muestren el comportamiento del modelo en función de un rango de valores seleccionados (por ejemplo, tamaño de N vs. tiempo, theta vs. tiempo, número de procesadores de la GPU vs. tiempo). Como un proceso adicional de validación, se habilitará un conjunto separado de características de GPUs utilizadas en el ámbito científico, buscando estimar su comportamiento y verificando si los modelos entrenados son generalizables a arquitecturas de hardware distintas a las utilizadas.

**Aporte Propio:** Scripts y recopilación exhaustiva de datos en múltiples GPUs, análisis y selección de parámetros específicos para *Bonsai*, diseño y ejecución del proceso de entrenamiento, ajuste de hiperparámetros, y desarrollo del software de visualización de modelos.

**Material Externo:** Herramientas de perfilado de NVIDIA CUDA, código base de Bonsai, y estructuras de entrenamiento recomendadas en trabajos de investigación afines.

# CAPÍTULO III

## MARCO METODOLÓGICO

Presentamos los pasos detallados para la captura, preprocesamiento y ensamblaje de los modelos de ejecución propuestos.

### 3.1 Recopilación, Análisis y Procesamiento

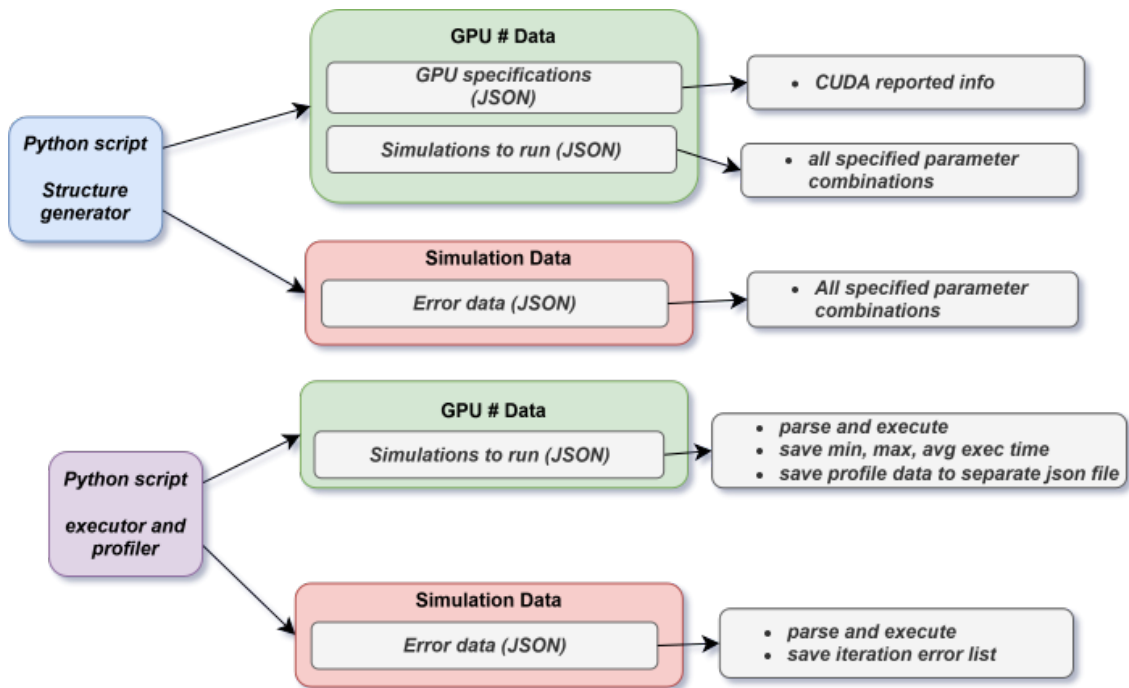


FIGURA 3.1: Primera fase: Recopilación de datos

### 3.1.1 *Testbed* de ejecución

Se realizará esta evaluación sobre 5 distintas configuraciones de hardware. La siguiente tabla muestra las siguientes GPUs que tendremos a disposición para la captura de métricas y perfilado.

GPU	C.C	Memory (MB)	Clock (MHz)	L2 Cache (KB)	SMs	Total Cores
GTX 1060 6GB	6.1	6065	1771	1536	10	1280
RTX 2070 SUPER	7.5	7790	1785	4096	40	5120
RTX 3060	8.6	11939	1777	2304	28	3584
RTX 3070	8.6	7877	1755	4096	46	5888
RTX 4060 Ti	8.9	7843	2565	32768	34	4352

TABLA 3.1: GPUs de Entrenamiento

### 3.1.2 Captura de características de hardware

Se extrae mediante un script las características de hardware de las GPU utilizando el API proporcionado por el lenguaje CUDA. Este obtendrá variables de Compute Capability, Compute Cores/SM, SM CUDA Cores, frequency, y otras que exponga el hardware de la GPU. Esta data será almacenada para su posterior evaluación el en preprocesamiento de los modelos ML, sirviendo como entrada para la caracterización de un conjunto de modelos de ejecución.

---

```

[
  {
    "Name": "NVIDIA GeForce RTX 3070",
    "Compute Capability": "8.6",
    "Total Memory (MB)": 7877,
    "Multiprocessors (SMs)": 46,
    "Max Threads Per SM": 1536,
    "Total Cores": 5888,
    "Warp Size": 32,
    "Max Threads Per Block": 1024,
    "Max Blocks Per SM": 16,
    "Shared Memory Per Block (KB)": 48,
    "Shared Memory Per SM (KB)": 100,
    "Registers Per Block": 65536,
    "Registers Per SM": 65536,
    "L1 Cache Size (KB)": 100,
    "L2 Cache Size (KB)": 4096,
    "Memory Bus Width (bits)": 256,
    "Memory Bandwidth (GB/s)": 448,
    "Clock Rate (MHz)": 1755,
    "Warps Per SM": 48,
    "Blocks Per SM": 16,
    "Half Precision FLOP/s": 9623,
    "Single Precision FLOP/s": 4811,
    "Double Precision FLOP/s": 4811,
    "Concurrent Kernels": 1,
    "Threads Per Warp": 32,
    "Global Memory Bandwidth (GB/s)": 448,
    "Global Memory Size (MB)": 7877,
    "L2 Cache Size": 4096,
    "Memcpy Engines": 2
  }
]

```

---

FIGURA 3.2: gpu\_info.json

### 3.1.3 *Plan* de ejecución y captura de parámetros de ejecución

Se programa un script de creación de ejecuciones, escribiendo archivos en formato .json especificando múltiples instancias de parámetros de ejecución, los cuales serán llenados conforme se ejecuten las pruebas de ejecución del sistema en prueba.

El siguiente archivo (runs.json), capturará el conjunto de métricas destinadas a la predicción de los modelos de tiempo de ejecución (min, max y promedio del muestreo), y una referencia al archivo de perfilado que será ejecutado. Este archivo se ejecuta para cada sistema (GPU) en el conjunto de prueba.

---

```

{
  "N": 100,
  "theta": 0.2,
  "dt": 6.25e-05,
  "I": 2,
  "exec_time_min": null,
  "exec_time_max": null,
  "exec_time_avg": null,
  "profiling": false
}
...

```

---

FIGURA 3.3: runs.json

El siguiente archivo (errors.json), captura las métricas destinadas al modelo de error de energía de la simulación (lista de error acumulada a través de la iteración). Este dato, al ser un error proveniente de la implementación de *Bonsai* y no variar en torno a la configuración de hardware utilizada, será ejecutada en la configuración de hardware de mayor disposición.

---

```

{
  "N": 100,
  "theta": 0.2,
  "dt": 6.25e-05,
  "I": 10000,
  "energy_error_list": null
}
...

```

---

FIGURA 3.4: error.json

Con el objetivo de obtener un conjunto de data substancial, necesaria para el posterior análisis y modelado, se establece una evaluación extensa de configuraciones de ejecución de *Bonsai*, planteando rangos para las variables de ejecución con las que se ejecutará la simulación.

N	theta	dt	I
$10^2 - 10^7$	0.2 - 0.8	$0.0625 * 10^{-3} - 0.0625 * 10^6$	2 ( <i>avg 10 samples</i> )

TABLA 3.2: Rango de variables de ejecución

Cabe resaltar, que el plan de ejecución del conjunto de datos de error de simulación tomará un muestreo ligeramente reducido del rango presentado, debido

N	theta	dt	I
$10^2 - 10^6$	0.2 - 0.8	$0.0625 * 10^{-3} - 0.0625 * 10^2$	$10k$ (1 sample)

TABLA 3.3: Rango de variables de precisión

a los elevados tiempos de ejecución que implica una prueba más extensa. De igual manera, al ser destinado a un modelo mucho mas simple, consideramos que la data recopilada será suficientemente substancial.

Esta data será almacenada de manera categorizada para su inicial análisis mediante gráficas para evaluar el comportamiento de las diferentes configuraciones del hardware evaluado. Las gráficas más relevantes y su análisis serán presentados en la fase de resultados. A su vez, esta será utilizada en las fases de preprocesamiento y finalmente entrenamiento de los modelos de predicción.

#### 3.1.4 Selección de Kernels a perfilar

A diferencia de otros trabajos evaluados durante la revisión de literatura, resulta necesario para nuestro caso actual realizar una selección inicial de *Kernels* de ejecución a perfilar mediante nuestro esquema de captura. Esto se debe a el uso de múltiples kernels durante cada iteración de *Bonsai* para cumplir distintas fases del cálculo, como se menciona en la documentación y diseño del algoritmo. Se realizará un preanálisis de este comportamiento mediante las herramientas de perfilado a utilizar, considerando pertinente buscar los 5 kernels de mayor representación al conjunto de perfilado en base a las 5 fases principales de Bonsai acorde a la implementación (*Sorting, Moving, Construction, Properties y Tree Transverse*).

La extracción de estos kernels se realiza en base al porcentaje de ejecución de cada kernel frente al tiempo total de ejecución de la simulación, procesando dicha data para parámetros de ejecución de valor significativo ( $N = 10^7$ ).



Cabe resaltar que sería necesario realizar este preanálisis desde cero en caso se busque implementar este esquema para otra simulación en GPU diferente a la evaluada, tanto para otras implementaciones N-Body como para otras ramas de la simulación científica.

### 3.1.5 Captura de métricas de perfilado

Se utilizan las herramientas *NVPROF* y *NSight Compute* para almacenar las métricas de perfilado de cada ejecución del código, filtrando la captura de Kernels sobre los determinados previamente. Estas nos brindarán, por ejemplo, las métricas de occupancy de los SM de la GPU, y otros conjuntos de características que evaluaremos durante el preprocesamiento para maximizar la correlación de nuestro modelo.

---

```
# profiling_{N}_{theta}_{dt}.json
{
  "ID": 0,
  "Process ID": 1273648,
  "Process Name": "bonsai2_slowdust",
  "Host Name": "127.0.0.1",
  "Kernel Name": "dev_approximate_gravity",
  "Context": 1,
  "Stream": 14,
  "Block Size": "(128, 1, 1)",
  "Grid Size": "(1472, 1, 1)",
  "Device": 0,
  "CC": 8.6,
  "Section Name": "Command line profiler metrics",
  "Metric Name": "dram_bytes_read.sum.per_second",
  "Metric Unit": "byte/second",
  "Metric Value": "3,353,672,316.38"
}
```

---

FIGURA 3.5: Ejemplo de perfilado para Nsight compute

Para casos en que sea necesario realizar el proceso de perfilado mediante NSight Compute, se realizará una conversión de características de la estructura base generada por NVPROF, con el objetivo de mantener un conjunto uniforme de características para los posteriores pasos a realizar.

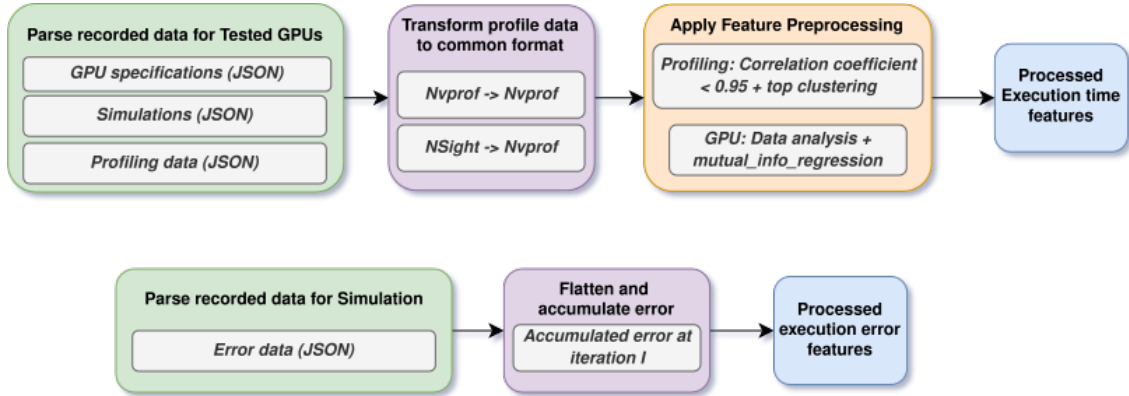


FIGURA 3.6: Segunda fase: Análisis y preprocesamiento

### 3.1.6 Construcción de *Dataset* completo

Una vez completados la captura de ejecución y perfilado, se genera un dataset final en formato .csv, tabulando todas las combinaciones de características de ejecución, características de la GPU que lo ejecuta, variables de perfilado medidas, y tiempos de ejecución obtenidos. En este paso se realiza la conversión de las variables de perfilado a la estructura NVPROF, y se realiza una transformación de columnas para generar columnas de perfilado separadas para cada uno de los 5 kernels establecidos, obteniendo finalmente la siguiente estructura:

sim params	gpu params	exec params	profile params (kernel_1)	...	profile params (kernel_N)
N, theta, dt..	frequency, cores...	exec_time_avg	elapsed_cycles_sm, active_cycles_pm...	...	elapsed_cycles_sm, active_cycles_pm..

TABLA 3.4: Estructura de dataset de ejecución

Respecto al conjunto de datos de precisión, se realiza genera un acumulado del error de precisión para cada iteración del dataset, obteniendo la siguiente estructura:

sim params	error param
N, theta, dt, I	accumulated_error

TABLA 3.5: Estructura de dataset de precisión

### 3.1.7 Análisis y Preprocesamiento de características para ejecución

#### 3.1.7.1. Características de Simulación

Se realizará una evaluación del comportamiento de cada característica de simulación ( $N$ ,  $\theta$ ,  $dt$ ) frente a la variable objetivo, evaluando el comportamiento mediante ejemplos de gráficas para verificar su comportamiento y establecer, de ser necesario, algún filtrado adicional para mitigar la presencia de *outliers* en el conjunto de datos. Estas gráficas serán agrupadas por GPUs en su visualización.

#### 3.1.7.2. Características de GPU

Debido a la importancia de estas características para la predicción de nuestro modelo, se establecerá una selección de variables de GPU para la predicción de los modelos, en base a un análisis del comportamiento cada característica a través de gráficas versus la variable objetivo, en conjunto a recomendaciones de trabajos relacionados y un filtrado mediante el scoring *mutual\_info\_regression* (ej. 5 características).

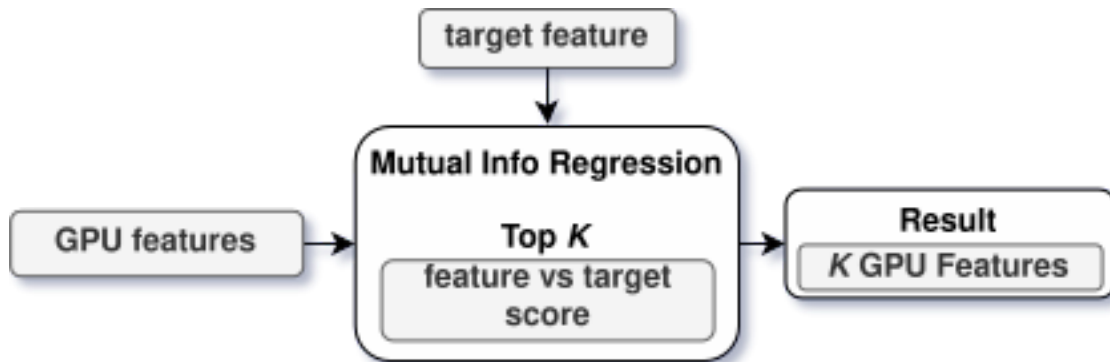


FIGURA 3.7: Esquema de selección mediante *mutual\_info\_regression*

Debido al reducido número de características, este proceso resulta suficiente para obtener un conjunto de características de entrada a partir de estas columnas.

### 3.1.7.3. Características de Perfilado

Debido a la considerable cantidad de métricas de perfilado capturadas a través de múltiples kernels, se establece un proceso de filtrado en base a correlación y clustering, con el objetivo de reducir y encontrar las características más significativas de este conjunto de datos, frente al tiempo de ejecución.

Se sigue el siguiente esquema:

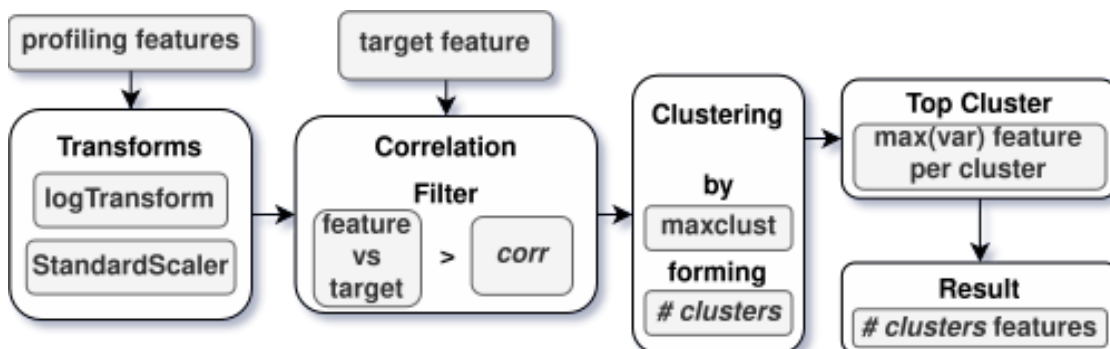


FIGURA 3.8: Esquema de selección de características de perfilado

El esquema propuesto realiza un filtrado en base a la correlación de cada característica de perfilado frente al tiempo de ejecución, filtrando solamente las características con correlación absoluta mayor a cierto valor (ej. 0.95). Con este filtrado inicial, se realiza un clustering jerárquico de parámetros en base a su similitud en un numero determinado de clusters de características (ej. 5 clusters). Sobre estos clusters, se realiza un análisis de varianza para escoger la variable más representativa de cada conjunto. Utilizando este esquema, se obtienen un conjunto reducido de características de perfilado para realizar las pruebas de nuestros modelos.

### **3.1.8 Análisis y Preprocesamiento de características para precisión**

#### **3.1.8.1. Características de Simulación**

Se realizará nuevamente una evaluación del comportamiento de cada característica de simulación ( $I$ ,  $N$ ,  $theta$ ,  $dt$ ) frente a la variable objetivo acumulada, evaluando el comportamiento mediante ejemplos de gráficas para verificar su comportamiento bajo diferentes configuraciones y establecer, de ser necesario, algún filtrado adicional para mitigar la presencia de *outliers* en el conjunto de datos. Estas gráficas serán agrupadas por diferentes características dependiendo de la variable evaluada, buscando presentar diferentes escenarios de su comportamiento.

#### **3.1.9 Características finales**

Las características seleccionadas para ambos modelos se almacenan para su posterior uso durante el entrenamiento, y se toman en consideración las observaciones y el filtrado adicional por característica de verse necesario para un correcto entrenamiento de los modelos.

## 3.2 Modelado de ejecución y precisión

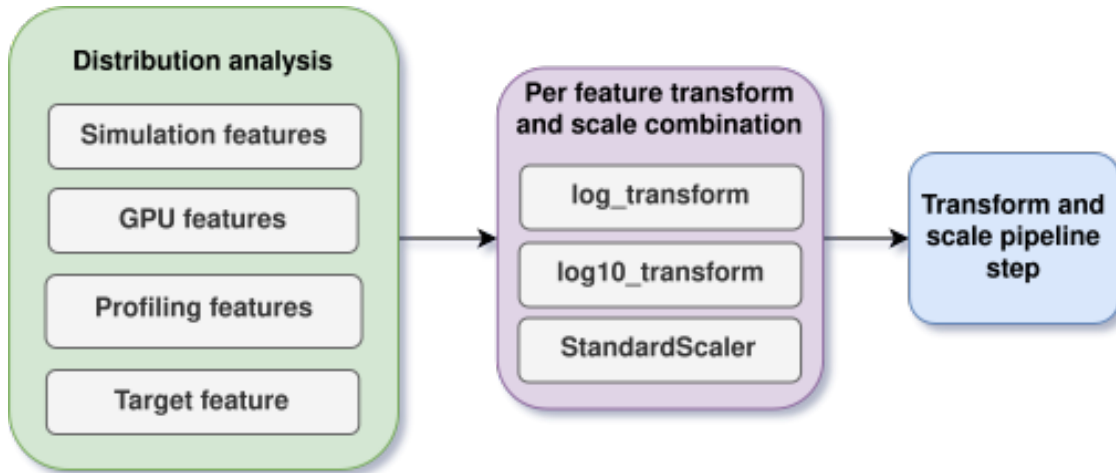


FIGURA 3.9: **Primera fase:** Análisis de distribución y transformaciones

### 3.2.1 Distribución de características

En base a las características seleccionadas para ambos modelos, se generarán histogramas para evaluar las distribuciones de valores de cada característica, tanto de entrada como de salida, con el fin de definir una secuencia de transformaciones para cada característica, con el objetivo de normalizar sus distribuciones de datos y facilitar el posterior entrenamiento de los modelos.

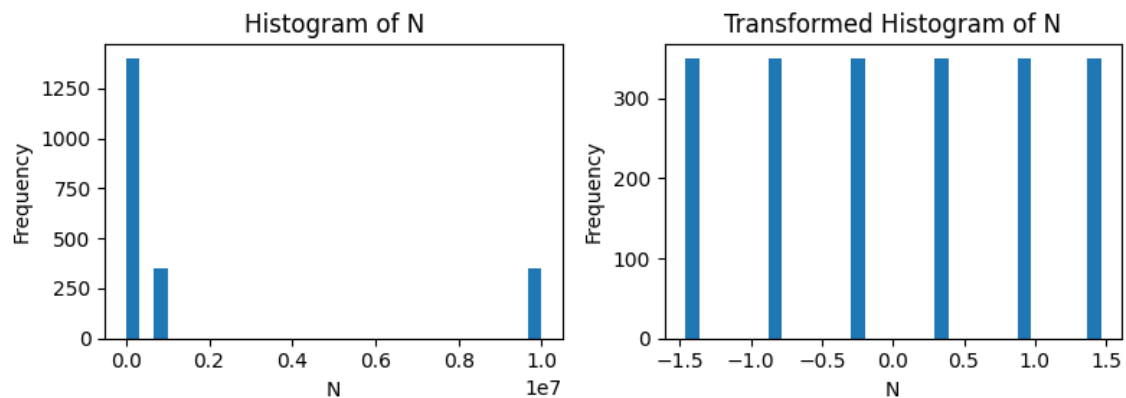


FIGURA 3.10: Ejemplo de distribución de N y subsecuente transformación normalizada

### 3.2.2 Transformación y escalado de características

Con el fin de mejorar el rendimiento de los modelos y reducir el error en las predicciones, se aplicarán transformaciones y escalado de características basados en las distribuciones evaluadas durante el preprocesamiento de los datos. Estas transformaciones están diseñadas para normalizar los datos, facilitar el entrenamiento de los modelos y mejorar la capacidad de generalización.

A partir de un conjunto de pruebas iniciales y el análisis de trabajos relacionados, se ha determinado el uso de las siguientes transformaciones para los datasets:

transform / scale
logTransform
log10Transform
StandardScaler

TABLA 3.6: Transformaciones a disposición

Las pruebas preliminares indicaron que el uso de transformadores mas robustos como el *QuantileTransformer* lograban obtener distribuciones de datos más

uniformes y valores de error MAPE más bajos durante el entrenamiento. Sin embargo, este enfoque limitaba significativamente la capacidad de interpolar y extrapolar resultados en los modelos, lo que afectaba la precisión en las predicciones. Por esta razón, se ha decidido no emplear el *QuantileTransformer* para preservar la capacidad de generalización de los modelos.

Estas transformaciones serán aplicadas como un paso inicial dentro del *pipeline* de cada modelo, asegurando que las características de entrada sean apropiadamente preprocesadas antes de su uso en el entrenamiento, tanto para las características de entrada como de salida, y a su vez revirtiendo las transformaciones de salida durante el cálculo del error MAPE y MAE.

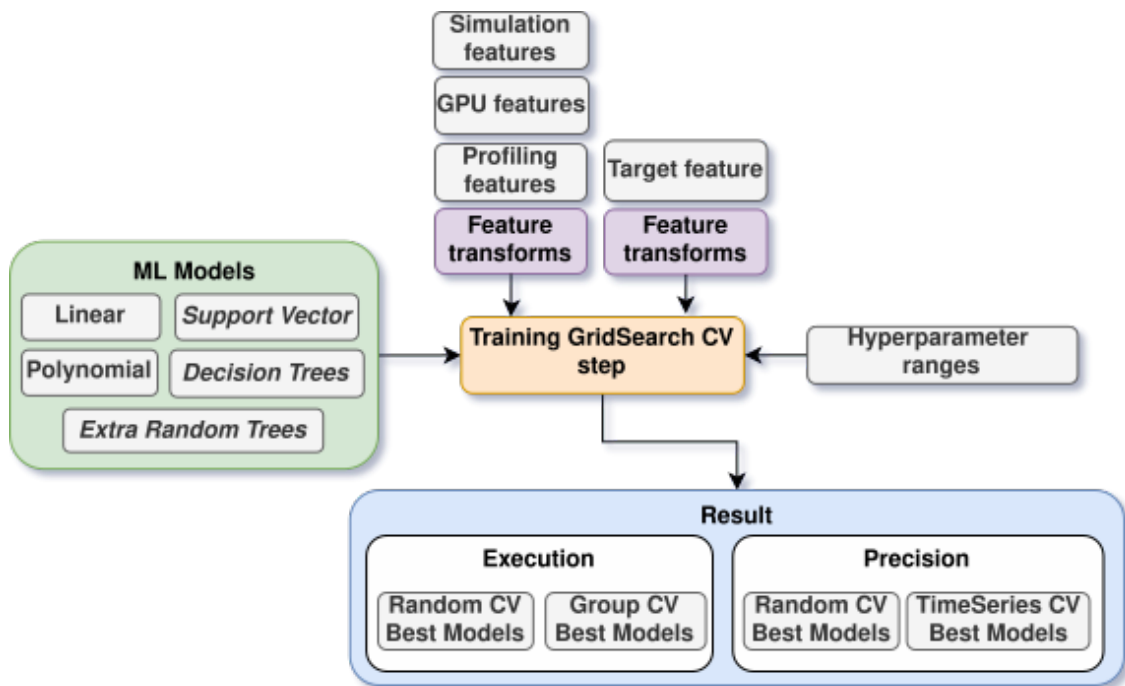


FIGURA 3.11: **Segunda fase:** Entrenamiento y selección de modelos



### 3.2.3 Entrenamiento mediante Cross-Validation

Se seguirá el siguiente esquema para el entrenamiento de nuestros modelos utilizando *cross validation* y búsqueda exhaustiva de hiperparámetros, aplicando las transformaciones y escalado anteriormente designadas para cada característica del conjunto de datos.

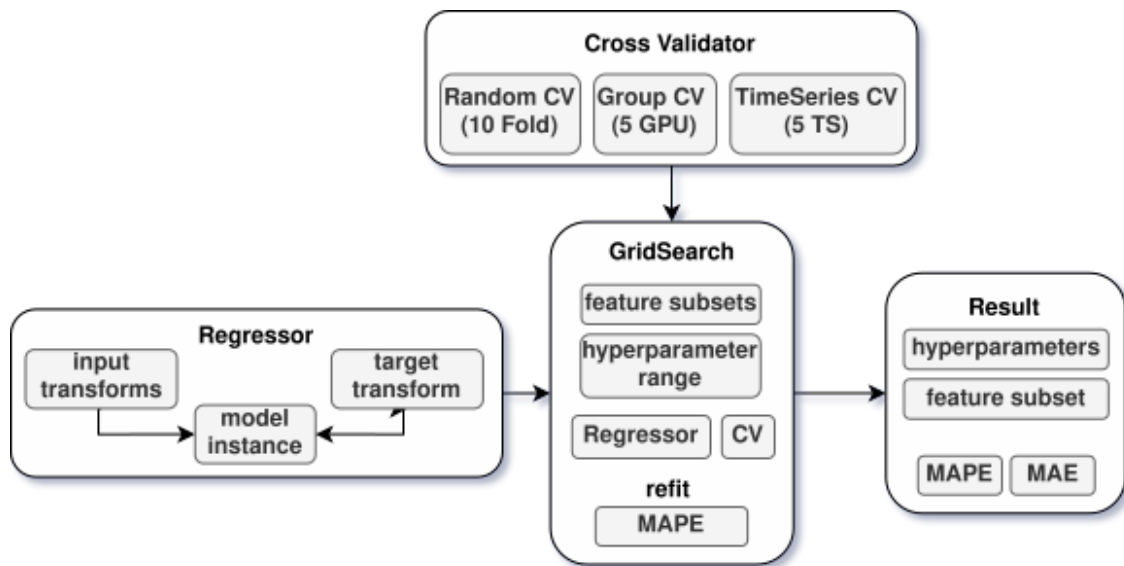


FIGURA 3.12: Esquema detallado de entrenamiento

El framework establecido plantea una búsqueda exhaustiva de hiperparámetros de modelo y de conjuntos de características, buscando minimizar el error MAPE evaluado a través de rondas de cross validation, establecidas de manera aleatoria o agrupadas a través de GPU (Ejecución) o de rangos de iteraciones (Precisión)

### 3.2.4 Modelos y rangos de hiperparámetros

Se establecen los siguientes modelos y sus respectivos rangos de hiperparámetros para entrenamiento mediante búsqueda exhaustiva:

**Huber Regressor:** Regresión lineal L2 robusta contra *outliers*.

FIGURA 3.13: **Huber**, hiperparámetros de entrenamiento

---

```
epsilon: [1, 1.35, 1.5, 1.75]
alpha: [0.00001, 0.0001, 0.001, 0.01, 0.1, 1]
```

---

**Polynomial Ridge Regressor:** Características polinomiales junto a regresión lineal L2.

FIGURA 3.14: **Polynomial Ridge**, hiperparámetros de entrenamiento

---

```
poly_degree: [2, 3, 4, 5, 6, 7]
ridge_alpha: [0.0001, 0.001, 0.01, 0.1]
```

---

**Support Vector Regressor:** Regresión mediante máquinas de soporte vectorial, utilizando kernel RBF.

FIGURA 3.15: **Support Vector**, hiperparámetros de entrenamiento

---

```
C: [0.0001, 0.001, 0.01, 0.1, 1, 10, 50, 100, 1000,
    2000]
epsilon: [0.00001, 0.0001, 0.001, 0.01, 0.1, 0.5, 1]
```

---

**Decision Tree Regressor:** Regresión mediante árboles de decisión.

FIGURA 3.16: **Decision Tree**, hiperparámetros de entrenamiento

---

```
max_depth: [2, 5, 10, 20, None]
min_samples_split: [2, 5, 10]
min_samples_leaf: [1, 2, 4]
max_features: [0.2, 0.4, 0.6, 0.8, 1.0]
```

---

**Extra Tree Regressor:** Regresión mediante resultados promedio de una serie de árboles de decisión aleatorios.

FIGURA 3.17: **Extra Tree**, hiperparámetros de entrenamiento

---

```
n_estimators: [50, 100, 200, 300, 500]
max_depth: [10, 20, 30, None]
min_samples_split: [2, 5, 10]
min_samples_leaf: [1, 2, 4]
max_features: [0.2, 0.4, 0.6, 0.8, 1.0]
```

---

### 3.2.5 Categorías de modelos a entrenar

#### 3.2.5.1. Ejecución

Cada modelo de regresión será entrenado y evaluado utilizando dos enfoques de *Cross Validation*. En el primero, se empleará *Random Cross Validation* con 10 *folds*, distribuyendo aleatoriamente los datos en conjuntos de entrenamiento y prueba. En el segundo, se aplicará *Grouped Cross Validation*, donde los datos se agruparán por las 5 GPUs utilizadas en el entrenamiento, dejando una GPU diferente en cada iteración para la validación. Este enfoque permitirá analizar la capacidad de los modelos para generalizar a hardware no observado previamente.

Además, se evaluarán dos subcategorías de modelos para explorar el impacto de diferentes características en las predicciones:

**Ejecución sin perfilado:** Se realizará el entrenamiento utilizando únicamente las características de simulación y especificaciones de hardware, sin incluir las métricas obtenidas del perfilado. Este enfoque se alinea con el objetivo final de proporcionar una herramienta capaz de estimar los tiempos de ejecución a partir de una descripción inicial de hardware y parámetros de simulación, sin necesidad de ejecutar previamente el código en la GPU. Las métricas de perfilado requieren una ejecución

directa para ser recopiladas, lo que las convierte en una solución poco práctica para escenarios donde no se tiene acceso al código o no se desea realizar una evaluación previa en el sistema objetivo.

**Ejecución con perfilado:** Se incluirán las métricas obtenidas del perfilado en el entrenamiento para evaluar su impacto en la precisión del modelo, especialmente en la capacidad de diferenciar configuraciones de hardware similares. Este enfoque permitirá analizar en que medida la incorporación de características de utilización de recursos y el comportamiento interno de la GPU contribuye a minimizar los errores de predicción. La comparación con los resultados de la ejecución sin perfilado permitirá identificar el grado de dependencia del modelo en dichas métricas y su influencia en la generalización a diferentes plataformas de hardware.

La comparación de ambos enfoques permitirá evaluar el impacto relativo de las métricas de perfilado y determinar la estrategia más efectiva para realizar predicciones en un contexto práctico.

### 3.2.5.2. Precisión

Cada modelo de regresión será entrenado y evaluado utilizando dos enfoques de *Cross Validation*, con el objetivo de capturar diferentes aspectos de la generalización de los modelos. En el primero, se empleará *Random Cross Validation* con 10 *folds*, distribuyendo aleatoriamente los datos en conjuntos de entrenamiento y prueba de manera uniforme en el espacio de datos. En el segundo, se aplicará *Time Series Cross Validation* sobre el parámetro  $I$  (iteración actual), donde se dividirán los datos en 5 bloques secuenciales en función del valor creciente de  $I$ . En cada iteración, los datos de entrenamiento incluirán únicamente iteraciones anteriores al conjunto de

validación. Este método simula escenarios donde se desea proyectar el comportamiento de la simulación para valores de  $I$  no observados previamente, evaluando la capacidad del modelo para extrapolar a iteraciones futuras.

### 3.2.6 Presentación y comparativa de resultados

Se presentarán los resultados obtenidos tras aplicar las metodologías de Cross Validation y búsqueda exhaustiva para obtener los modelos finales, diferenciando entre los enfoques de ejecución con y sin perfilado, así como los enfoques de precisión basados en  $I$ .

#### 3.2.6.1. Ejecución

Una tabla resumirá el error MAPE y MAE obtenido por cada modelo final, comparando los resultados para los métodos de *Random Cross Validation* y *GPU Cross Validation*. Esto permitirá evaluar el desempeño general de cada modelo bajo configuraciones aleatorias y agrupadas por GPU. Una tabla auxiliar presentará un desglose detallado del error MAPE y MAE para cada GPU en el enfoque de *GPU Cross Validation*, analizando las diferencias individuales en el desempeño de cada modelo entrenado. Esta información será presentada tanto para los modelos entrenados sin perfilado como para los que incluyen variables de perfilado.

#### 3.2.6.2. Precisión

Una tabla resumirá el error MAPE y MAE de los modelos finales para ambos enfoques de *Cross Validation*: *Random Cross Validation* y *Time Series Cross Validation*, enfocándose en la capacidad predictiva de cada modelo. Una tabla auxiliar mostrará un desglose del error MAPE y MAE para cada bloque secuencial basado en

valores crecientes de  $I$ , evaluando el desempeño de los modelos en la extrapolación hacia iteraciones futuras.

Regressor	CV 1		CV 2	
	MAPE	MAE	MAPE	MAE
Model 1	# %	#	# %	#
Model 2	# %	#	# %	#

TABLA 3.7: Ejemplo de resultado principal

Fold	Model 1		Model 2	
	MAPE	MAE	MAPE	MAE
Fold 1	# %	#	# %	#
Fold 2	# %	#	# %	#

TABLA 3.8: Detalle de resultado auxiliar

Estas tablas proporcionarán una visión cuantitativa y comparativa del desempeño de los modelos, destacando la influencia de las variables de perfilado y la robustez de los modelos en diferentes escenarios de validación.

### 3.3 Software y evaluación de proyección

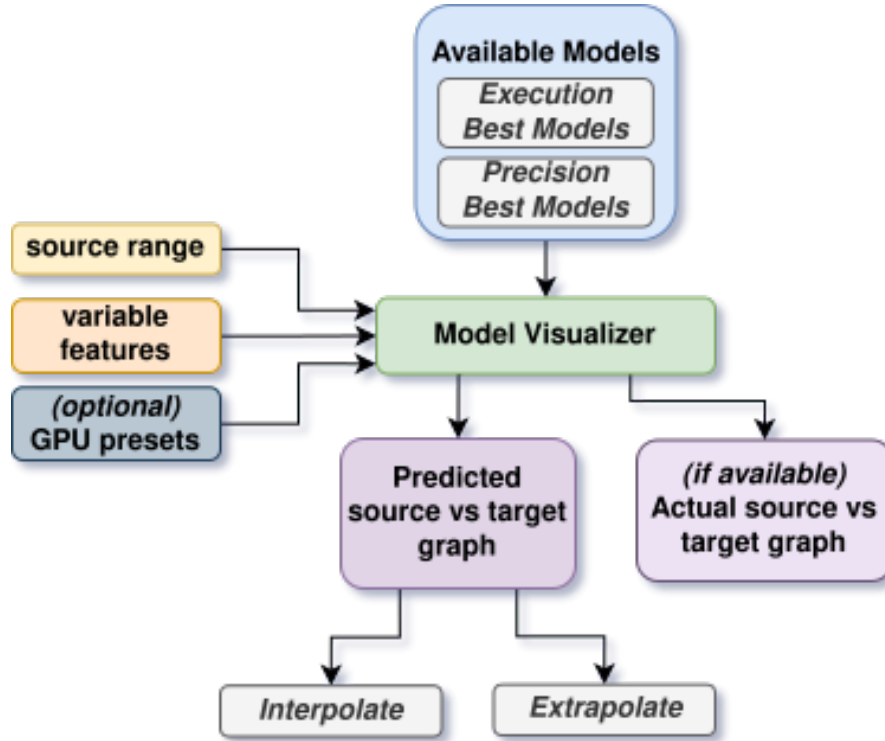


FIGURA 3.18: **Primera fase:** Implementación del visualizador y salidas esperadas

#### 3.3.1 Overview de Implementación

El software desarrollado tiene como objetivo principal proporcionar una herramienta interactiva y flexible para la evaluación de los modelos entrenados, permitiendo realizar proyecciones sobre tiempos de ejecución y errores acumulados en simulaciones *N-Body*. La implementación fue realizada utilizando *Python*, empleando la librería *Streamlit* para construir una aplicación web práctica y fácil de utilizar.

El sistema está diseñado para ser ejecutado tanto localmente como en un servidor remoto, permitiendo su acceso mediante una dirección online. Las visualizaciones gráficas se generan utilizando *matplotlib* y se visualizan como elementos

interactivos HTML en el navegador, ofreciendo gráficos dinámicos que permitan explorar los resultados de los modelos.

### 3.3.2 Carga y Selección de Modelos

El software integra un mecanismo que permite cargar los modelos resultantes del proceso de entrenamiento. Estos modelos están categorizados según su finalidad, ya sea para predicción de tiempos de ejecución o para estimación de errores de simulación.

Mediante un conjunto de desplegados interactivos, el usuario puede seleccionar el modelo a evaluar, el cual mostrará su error MAPE como una métrica de referencia de su desempeño.

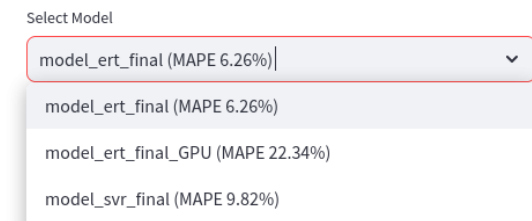


FIGURA 3.19: Selección de modelo

La arquitectura del software ajusta dinámicamente los parámetros de entrada y salida requeridos por cada modelo, habilitando los campos necesarios para que el usuario realice estimaciones personalizadas.

### 3.3.3 Selección de Variable Principal

La herramienta permite al usuario seleccionar una variable principal que servirá como eje de análisis en las proyecciones gráficas. Este enfoque sigue el formato *Variable Principal vs Variable Objetivo*, donde la variable objetivo corresponde al



tiempo promedio para los modelos de ejecución o al error acumulado para los modelos de precisión. La selección de la variable principal facilita la exploración detallada de diferentes escenarios de simulación y permite analizar cómo los cambios en un parámetro específico afectan el comportamiento del sistema.

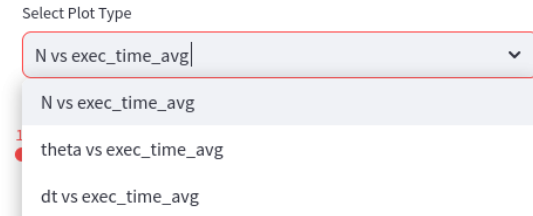


FIGURA 3.20: Selector de variable principal

### 3.3.4 Parámetros de Variable Principal y Características de Entrada

El sistema incluye un selector que permite definir los rangos de estimación para la variable principal seleccionada, permitiendo explorar un rango amplio o específico de valores para realizar predicciones detalladas. Además, se proporcionan campos de entrada numérica para que el usuario ingrese manualmente las características adicionales necesarias para completar las estimaciones, permitiendo evaluar diversas configuraciones de características de *hardware* y simulación.

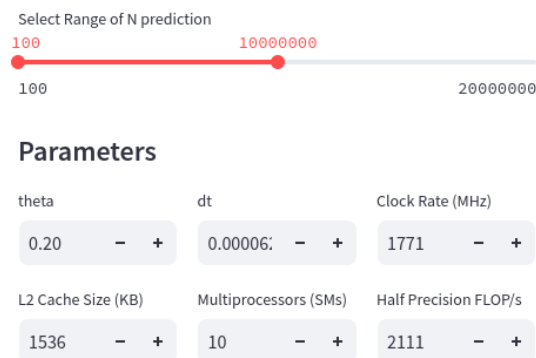


FIGURA 3.21: Selección de parámetros de simulación y GPU

### 3.3.5 Presets de GPU

Para simplificar el proceso de configuración de las características de hardware, se implementa un selector de *GPUs* por defecto, que permite cargar automáticamente las características correspondientes de una GPU utilizadas durante el entrenamiento. Esta funcionalidad resulta particularmente útil para realizar análisis comparativos entre diferentes GPUs entrenadas y proyectar su comportamiento en base a configuraciones conocidas.

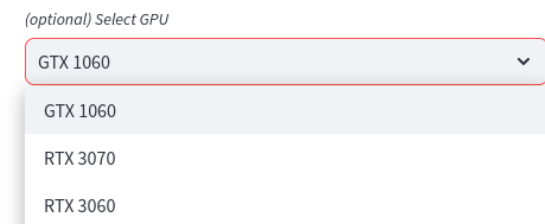


FIGURA 3.22: Selección de preset de GPU

### 3.3.6 Salidas Esperadas del Visualizador

#### 3.3.6.1. Gráfica de Estimación

El visualizador permite generar gráficos que muestran la relación entre la variable principal seleccionada y la variable objetivo. Estos gráficos son dinámicos y se actualizan en función de los parámetros de hardware y ejecución definidos por el usuario. El rango de la variable principal es ajustable, a la vez que los parámetros de simulación y hardware, lo que permite explorar tanto la interpolación como la extrapolación de los modelos seleccionados. Este análisis gráfico es esencial para comprender el comportamiento proyectado del sistema y evaluar la capacidad predictiva del modelo.

### 3.3.6.2. Gráfica del Dataset

En los casos en que los parámetros seleccionados coincidan con datos existentes en el dataset original, el visualizador genera una curva secundaria que sirve como referencia. Esta comparación directa entre los valores estimados por el modelo y los datos reales facilita la validación de los modelos y proporciona una métrica visual adicional para evaluar su precisión.

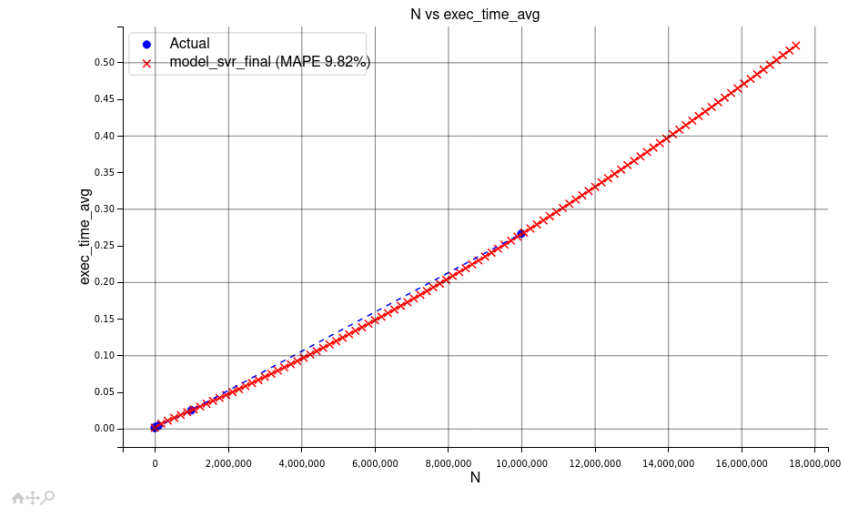


FIGURA 3.23: Ejemplo de gráfica estimada y existente para ejecución

### 3.3.7 Predicciones sobre GPUs Evaluadas

En esta sección, se analizará el comportamiento de las configuraciones de hardware utilizadas durante el proceso de entrenamiento de los modelos. Para ello, se generarán predicciones específicas que permitan observar tanto la interpolación como la extrapolación de los modelos en función de las características de hardware y parámetros de simulación ingresados. Esto permitirá determinar la robustez y generalización de los modelos al enfrentar escenarios diversos o configuraciones no vistas previamente.

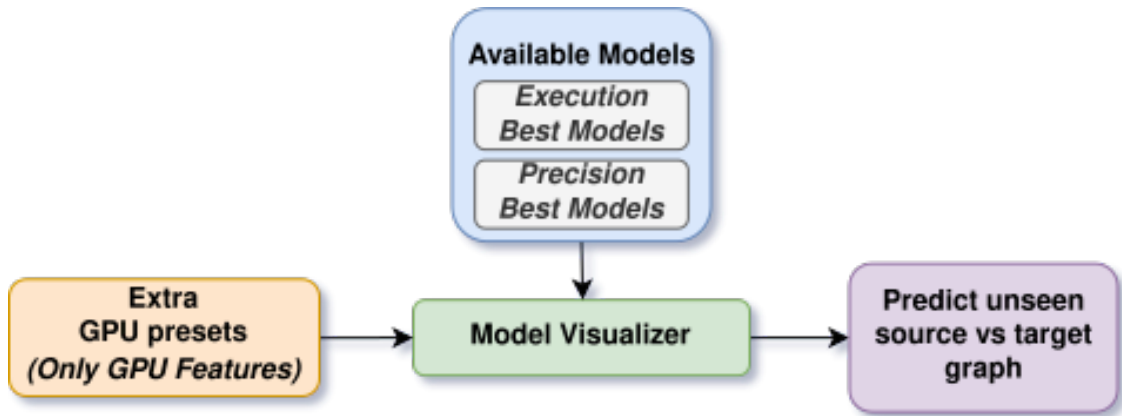


FIGURA 3.24: **Segunda fase:** Validación de GPUs fuera de alcance

### 3.3.8 Presets Adicionales de GPU

Se incluirán en el software características de hardware correspondientes a GPUs que no estuvieron disponibles durante el proceso de entrenamiento, con el objetivo de analizar el comportamiento de los modelos en hardware previamente no evaluado. Estos presets adicionales permitirán explorar cómo se comportan los modelos en escenarios que simulan configuraciones de hardware avanzadas, como aquellas utilizadas en servidores de alto rendimiento y sistemas HPC (High-Performance Computing).

GPU	Clock Rate (MHz)	Multi processors (SMs)	L2 Cache Size (KB)	Half Precision FLOP/s	Single Precision FLOP/s	Double Precision FLOP/s
Tesla T4	1590	40	4096	7581	3790	3790
Tesla P100	1328	56	4096	8865	4432	4432
A100	1065	108	81920	13711	6855	6855
A100 x 8	1065	108	81920 * 8	13711 * 8	6855 * 8	6855 * 8
A100 x 4320	1065	108	81920 * 8 * 540	13711 * 8 * 540	6855 * 8 * 540	6855 * 8 * 540

TABLA 3.9: GPUs fuera de alcance

Las GPUs seleccionadas para esta etapa incluyen la Tesla T4, Tesla P100, y A100, cuyas especificaciones fueron obtenidas a partir de instancias de Google Colab. Además, se incorporarán características de hardware escaladas del supercomputador

Selene [25], un sistema compuesto por 540 clústeres, cada uno equipado con 8 GPUs A100. Para representar este hardware, se realizará un escalado de los parámetros *L2 Cache* y *FLOP/s*, tanto a nivel de clúster como del sistema completo, proporcionando un rango de comparación muy por encima del entrenamiento realizado en nuestros modelos.

### 3.3.9 Predicciones sobre GPUs No Evaluadas

En esta etapa final, se realizarán predicciones utilizando los presets de hardware adicionales previamente definidos, con el propósito de evaluar el desempeño de los modelos en GPUs y configuraciones de hardware que no formaron parte del conjunto de entrenamiento. Este análisis permitirá explorar cómo los modelos enfrentan escenarios nuevos, destacando su capacidad para realizar interpolaciones precisas y, particularmente, para extrapolar en configuraciones más extremas. El estudio se extiende a un caso extremo al incluir comparativas con las configuraciones de hardware del supercomputador Selene. Esto permite evaluar si los modelos son capaces de proyectar resultados coherentes incluso en configuraciones muy fuera del rango de datos originales.

# CAPÍTULO IV

## RESULTADOS

### 4.1 U1. Recopilación, Análisis y Perfilado

Se presentan los resultados de la siguiente fase de nuestra experimentación.

#### 4.1.1 Selección de Kernels

Realizando la extracción de los kernels de mayor importancia a partir de pruebas mediante  $N = 10^7$ , se obtiene los siguientes kernels a perfilar:

Name	exec_percentage
dev_approximate_gravity	98.72 %
correct_particles	0.15 %
cl_build_key_list	0.12 %
compute_leaf	0.11 %
predict_particles	0.08 %

TABLA 4.1: Top 5 kernels

Los kernels seleccionados logran hacer referencia con las secciones principales de *Bonsai*: *predict\_particles* y *dev\_approximate\_gravity* siendo parte de las fases de *Properties* y *Tree Transverse* de la simulación, *cl\_build\_key\_list* de la fase de *Sorting* de partículas durante la iteración, *compute\_leaf* como parte de *Construction* de la estructura *Octree* interna, y *correct\_particles* de la fase de *Moving* de la reestructuración de las partículas en el árbol. Se determina de esta manera que las características

de perfilado que se obtendrán a partir de estos kernels serán representativas del comportamiento de la simulación.

#### 4.1.2 Datasets finales

Se presenta las dimensiones de los datasets generados para el análisis y entrenamiento de nuestros modelos.

##### 4.1.2.1. Ejecución (2100 samples)

El proceso de adquisición de datos de ejecución, tras el proceso de estandarización para las 5 configuraciones de hardware y perfilado, obtiene las siguientes dimensiones de características de entrada:

sim params	gpu params	profile params
3 features	29 features	635 features

TABLA 4.2: Características completas de dataset de ejecución

##### 4.1.2.2. Precisión (1000100 samples)

El proceso de adquisición de datos de precisión, una vez reestructurado y calculado el acumulado de errores de energía a través de cada iteración, obtiene las siguientes dimensiones de características de entrada:

sim params
4 features

TABLA 4.3: Características completas de dataset de ejecución

### 4.1.3 Análisis y Preprocesamiento de Características de Ejecución

Se presenta el análisis y resultados de preprocesamiento de las características del conjunto de datos de Ejecución.

#### 4.1.3.1. N vs tiempo de simulación

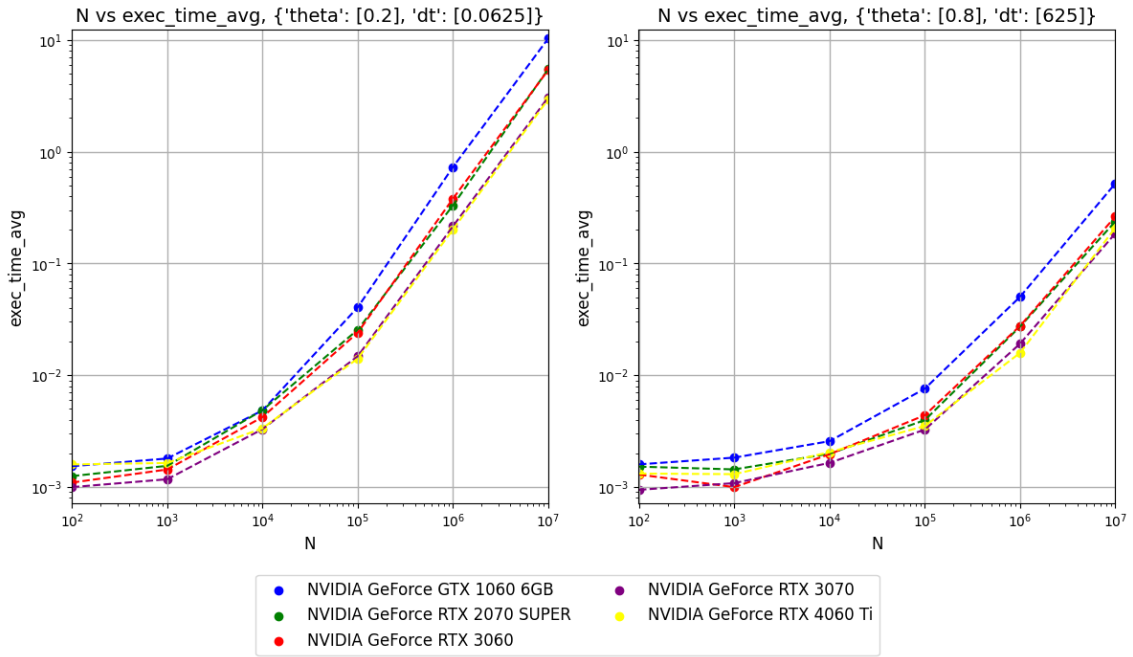


FIGURA 4.1: Curva de N vs tiempo de simulación, agrupado por GPU (escala log)

En las gráficas presentadas, se observan valores estables para todo el rango de  $N$ , con variaciones esperadas al modificar los parámetros  $\theta$  y  $dt$ . Además, las visualizaciones destacan las diferencias en el comportamiento entre las distintas GPUs evaluadas, especialmente en configuraciones con  $N = 10^7$ . La GPU con menores características (GTX 1060) presenta los tiempos de ejecución más altos, mientras que GPUs más avanzadas, como la RTX 3070 y la RTX 4060 Ti, muestran los tiempos más bajos. Un análisis más detallado revela que, en ciertas configuraciones de  $N$



y parámetros, GPUs como la RTX 3060 logran reducir sus tiempos de simulación por debajo de modelos superiores. Este fenómeno podría atribuirse a características específicas de su hardware, como el tamaño de la memoria, en el que supera a otras GPUs evaluadas.

#### 4.1.3.2. $\theta$ vs tiempo de simulación

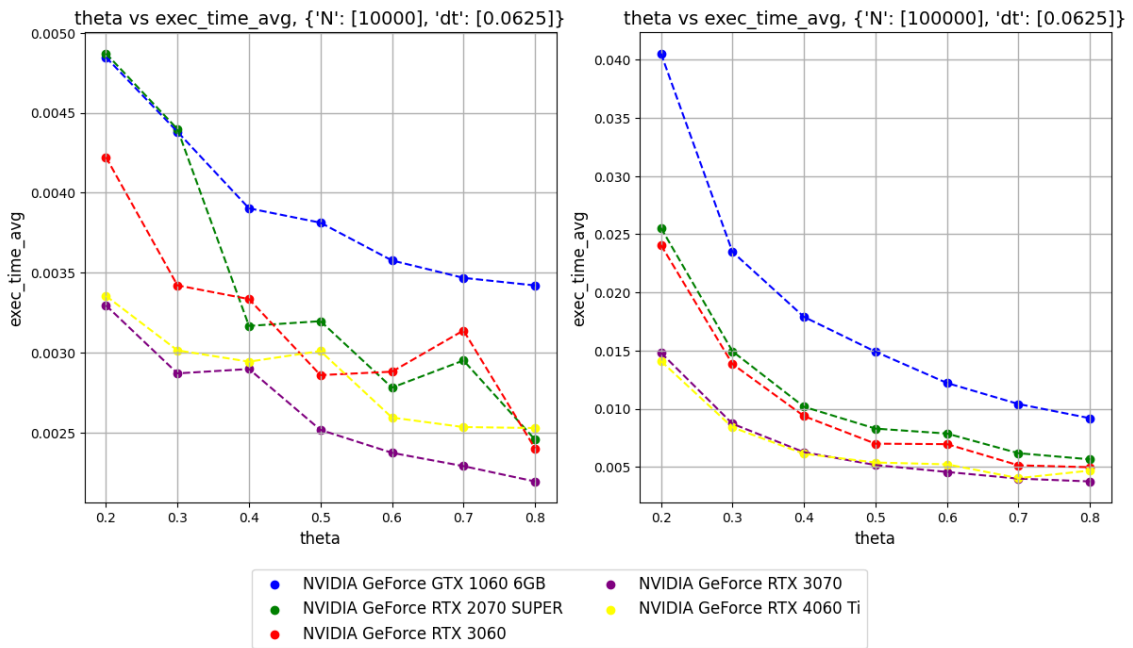


FIGURA 4.2: Curva de  $\theta$  vs tiempo de simulación, agrupado por GPU

En las visualizaciones de  $\theta$ , se observa una considerable presencia de ruido e inestabilidad en los datos para valores de  $N$  menores a  $10^5$ . Sin embargo, para  $N$  superiores, este comportamiento se estabiliza en una curva decreciente, lo que es consistente con la teoría del algoritmo *Barnes-Hut*, que predice una reducción en los tiempos de ejecución al aumentar el parámetro de aproximación  $\theta$ . Respecto a las diferencias por GPU, se confirma lo esperado: las GPUs más recientes y con características superiores, como la RTX 3070 y RTX 4060 Ti, mantienen los menores tiempos de ejecución.

#### 4.1.3.3. $dt$ vs tiempo de simulación

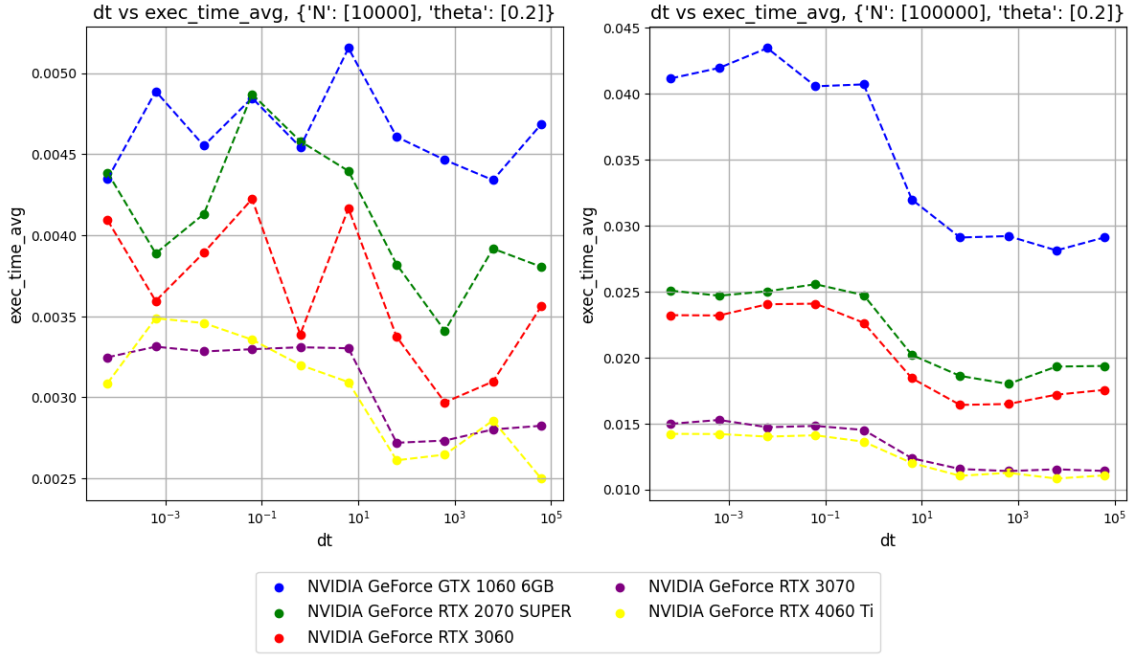


FIGURA 4.3: Curva de  $dt$  vs tiempo de simulación, agrupado por GPU (escala log)

El análisis de  $dt$  muestra nuevamente un notable ruido en los datos para valores de  $N$  menores a  $10^5$ , dificultando la identificación de tendencias estables en estos casos. Para  $N$  superiores, los datos se estabilizan y revelan un comportamiento logarítmico respecto a  $dt$ , indicando que los tiempos de simulación tienden a aumentar de forma más lenta a medida que el paso temporal crece. Las diferencias entre GPUs siguen el patrón observado en las visualizaciones de  $N$  y  $\theta$ , con las GPUs de mayores características logrando los menores tiempos de simulación.

#### 4.1.3.4. Observaciones Finales sobre características de simulación

Se denotan inconsistencias respecto al comportamiento de las variables para valores de  $N$  menores a  $10^5$ . Aunque estas inconsistencias puedan dificultar el entrenamiento y la representación de los modelos, se considera importante mantener el rango completo de estos parámetros, ya que  $N$  es la variable más importante a predecir mediante nuestros modelos de tiempos de ejecución.

#### 4.1.3.5. Características de GPU vs Tiempo de Simulación

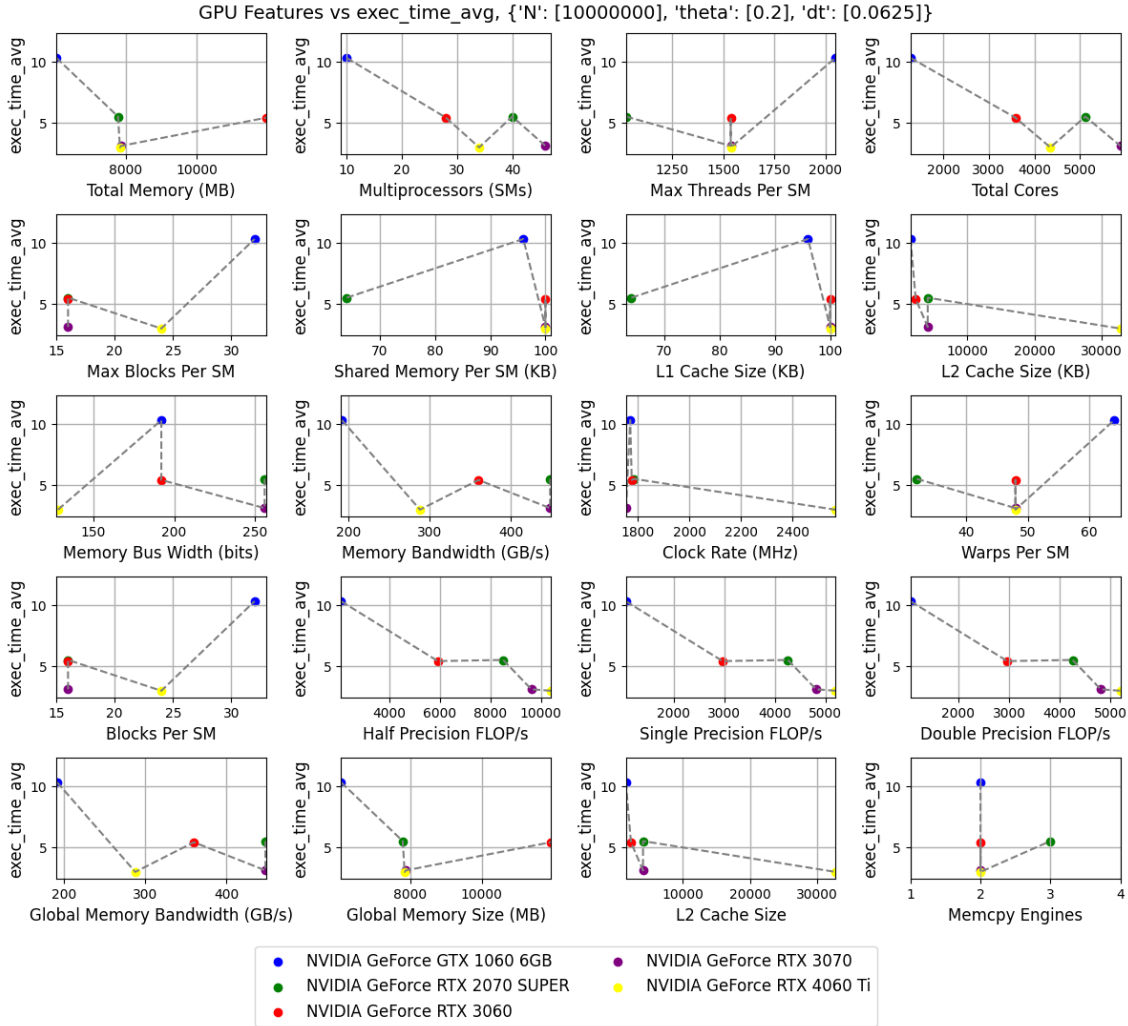


FIGURA 4.4: Características de GPU para N, theta y dt fijo

Se analizan las características de las GPUs y su influencia sobre el tiempo de ejecución de la simulación. El objetivo es identificar características que no generen superposición de tiempos de ejecución para los mismos valores. Algunos ejemplos negativos incluyen *Memcpy Engines* y *Memory Bus Width (bits)*.

Entre las variables con un comportamiento más representativo se destacan los *Half*, *Single* y *Double Precision FLOPs* de cada GPU. Trabajos relacionados recomiendan *L2 Cache Size (KB)* y *Multiprocessors (SMs)* como características representativas de las GPUs. Además, se considera *Clock Rate (MHz)* como una variable adicional que podría explicar diferencias de desempeño en simulaciones, como las observadas en la RTX 4060 Ti, que logra menores tiempos de ejecución a pesar de tener menos multiprocesadores que GPUs de mayor gama, debido a su *Clock Rate* significativamente superior al resto.

A su vez, tras realizar un análisis mediante *mutual information regression* limitado a las 5 características principales, se obtuvo el siguiente resultado:

GPU Features
Multiprocessors (SMs)
Total Cores
Half Precision FLOP/s
Single Precision FLOP/s
Double Precision FLOP/s

TABLA 4.4: Características de GPU mediante `mutual_info_regression`

Con base en los conjuntos de análisis presentados, se determinó el siguiente conjunto de características finales de GPU:

Final GPU Features
Clock Rate (MHz)
L2 Cache Size (KB)
Multiprocessors (SMs)
Half Precision FLOP/s
Single Precision FLOP/s
Double Precision FLOP/s

TABLA 4.5: Características finales de GPU

Es importante destacar que el proceso de entrenamiento llevará a cabo una búsqueda exhaustiva con diferentes combinaciones de características de GPU. Esto permitirá determinar cuáles de ellas logran representar mejor las configuraciones de hardware para cada clase de modelo evaluado.

#### 4.1.3.6. Características de perfilado vs tiempo de simulación

Sobre la etapa de preprocesamiento para el conjunto de datos del modelo de tiempo de simulación, y debido a la extensa cantidad de características a evaluar, se realiza inicialmente un análisis de correlación *Spearman*, seleccionando solamente las características con coeficientes de correlación mayores a **0.95** absoluto, frente a la variable objetivo (`exec_time_avg`). Esto genera una reducción de características iniciales de perfilado de **656** a **138** columnas.

Esto es seguido por el *Clustering* jerárquico de las 138 características filtradas, categorizando estas columnas a través de 5 diferentes conjuntos mediante el criterio *maxclust*. Sobre estos 5 conjuntos, se selecciona la característica de mayor varianza dentro de su propio conjunto como la más representativa, obteniendo finalmente **5** características de perfilado finales a utilizar durante el entrenamiento de esta clase de modelos.

Profiling Features
<code>gld_transactions</code> ( <code>dev_approximate_gravity</code> )
<code>thread_inst_executed</code> ( <code>dev_approximate_gravity</code> )
<code>inst_compute_ld_st</code> ( <code>correct_particles</code> )
<code>active_warps_pm</code> ( <code>correct_particles</code> )
<code>global_store_requests</code> ( <code>correct_particles</code> )

TABLA 4.6: Características finales de Perfilado mediante correlación y clustering

#### 4.1.4 Análisis y Preprocesamiento de Características de Precisión

En esta sección, se presentan los análisis realizados y los resultados obtenidos durante el preprocesamiento de las características del conjunto de datos de precisión.

##### 4.1.4.1. I vs Error Acumulado

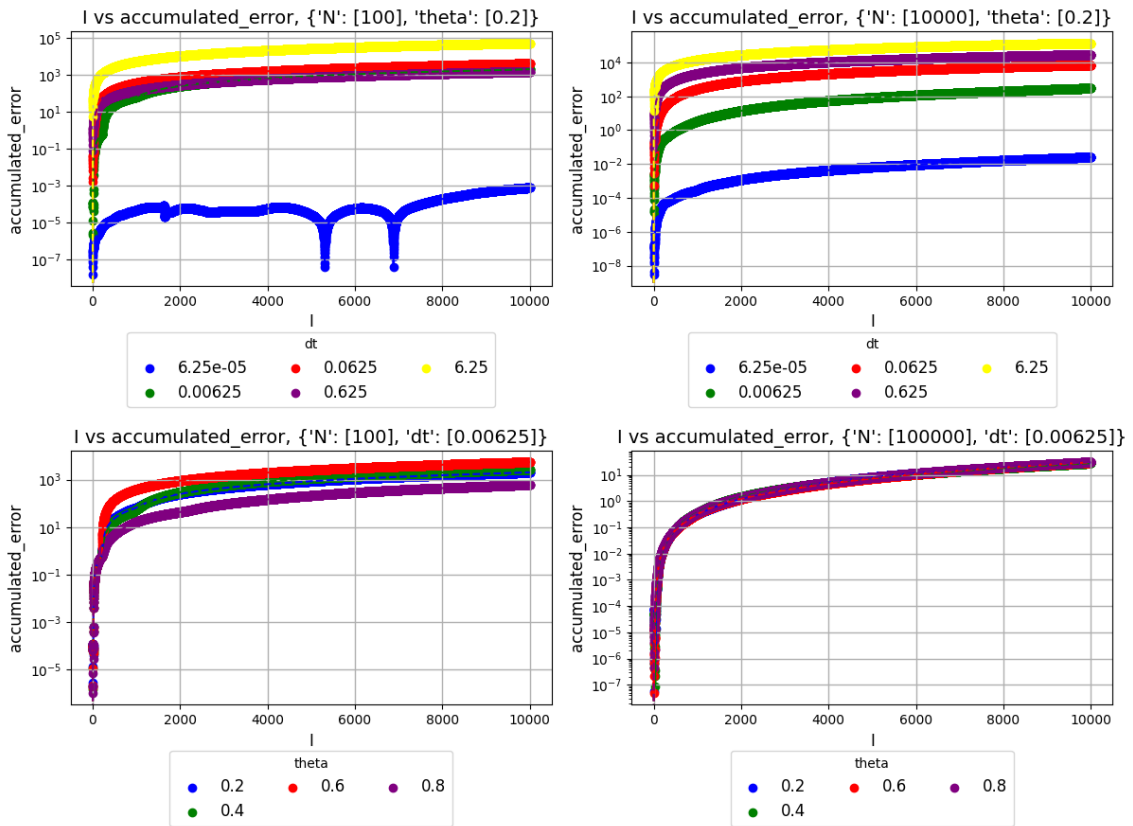


FIGURA 4.5: Curva de Iteraciones vs error acumulado, agrupado por  $dt$  y  $\theta$  (escala log)

Respecto a las curvas de  $I$ , se observa inicialmente un comportamiento inconsistente para  $N^2$  y valores muy pequeños de  $dt$ . A través del rango de  $I$ , las curvas agrupadas por  $dt$  muestran una creciente separación conforme aumenta  $N$ . Por otro lado, las curvas agrupadas por  $\theta$  tienden a converger entre sí a medida que

$N$  incrementa. Se visualiza un comportamiento exponencial de  $I$  respecto al error de simulación, con un crecimiento rápido que posteriormente satura su incremento. Este comportamiento se ve principalmente afectado por  $dt$ .

#### 4.1.4.2. $N$ vs Error Acumulado

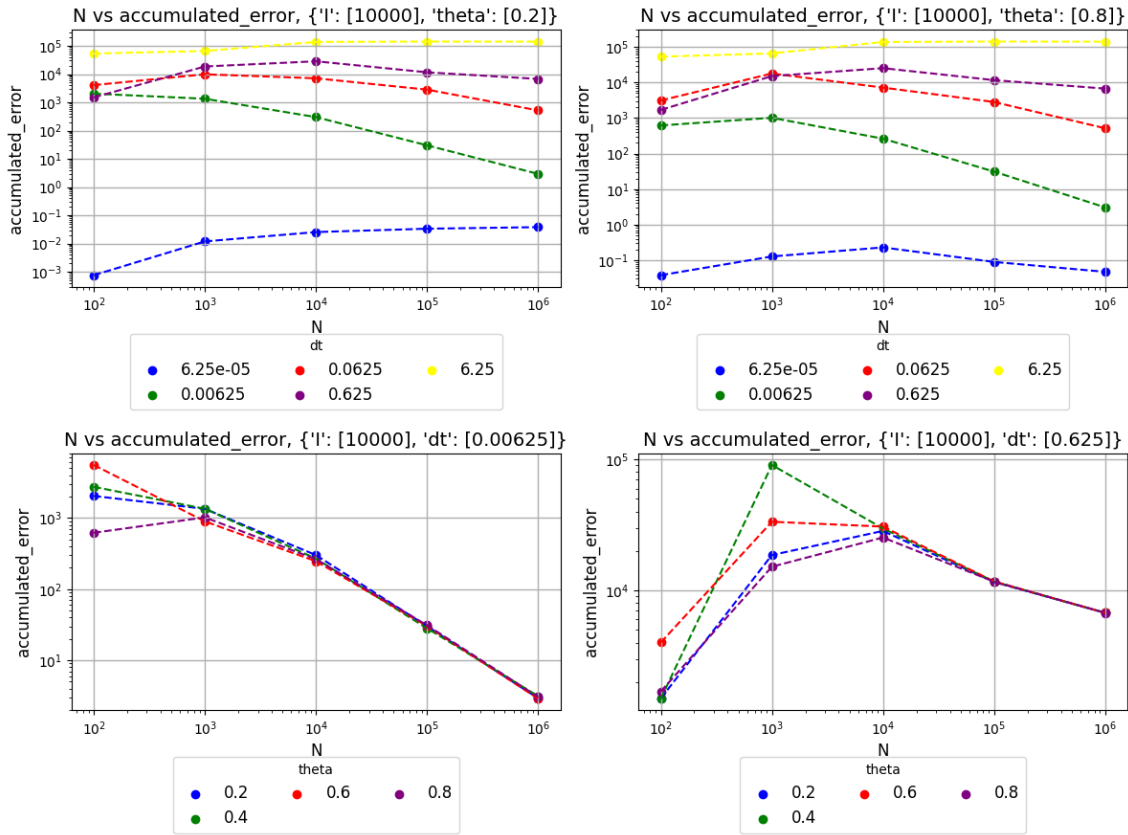


FIGURA 4.6: Curva de  $N$  vs error acumulado, agrupado por  $dt$  y  $\theta$  (escala log)

Respecto a  $N$ , y agrupando por  $dt$ , se observa variabilidad para  $dt$  muy pequeños y  $N = 10^2$ . El resto de las curvas se mantienen estables. Agrupando por  $\theta$ , se observa que valores pequeños de  $dt$  causan que el error disminuya conforme crece  $N$ . Para valores grandes de  $dt$ ,  $N = 10^2$  presenta un error muy bajo, incrementando drásticamente en  $N = 10^3$  y comenzando a reducirse para valores superiores de  $N$ .



#### 4.1.4.3. Theta vs Error Acumulado

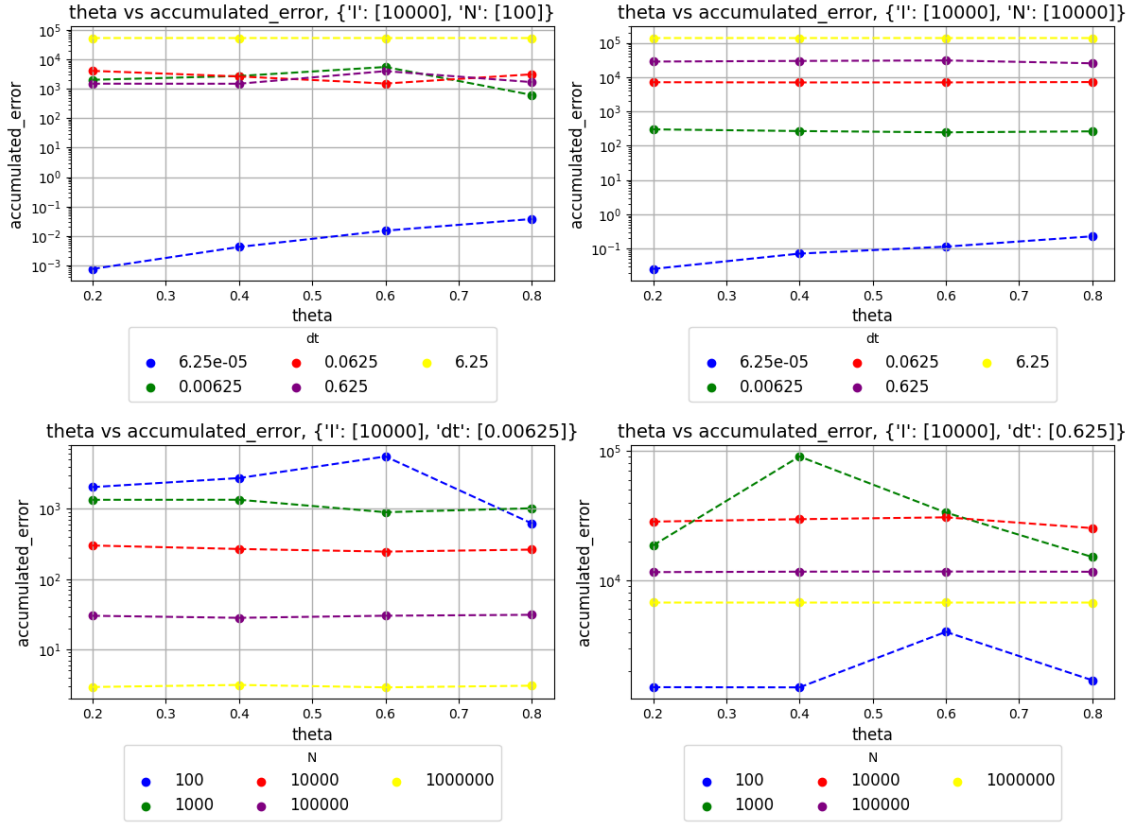


FIGURA 4.7: Curva de theta vs error acumulado, agrupado por  $dt$  y  $N$  (escala log)

Agrupando por  $dt$ , este varía considerablemente en valores superiores a  $6.25 \times 10^{-5}$  cuando  $N$  es pequeño. Para valores superiores de  $N$ , el comportamiento se mantiene mucho más estable. Agrupando por  $N$ , se observa que para  $N = 10^2$  y  $N = 10^3$ , el comportamiento varía considerablemente según el tamaño de  $dt$ . En valores superiores de  $N$ , el comportamiento se mantiene incluso tras variar  $dt$ .

#### 4.1.4.4. dt vs Error Acumulado

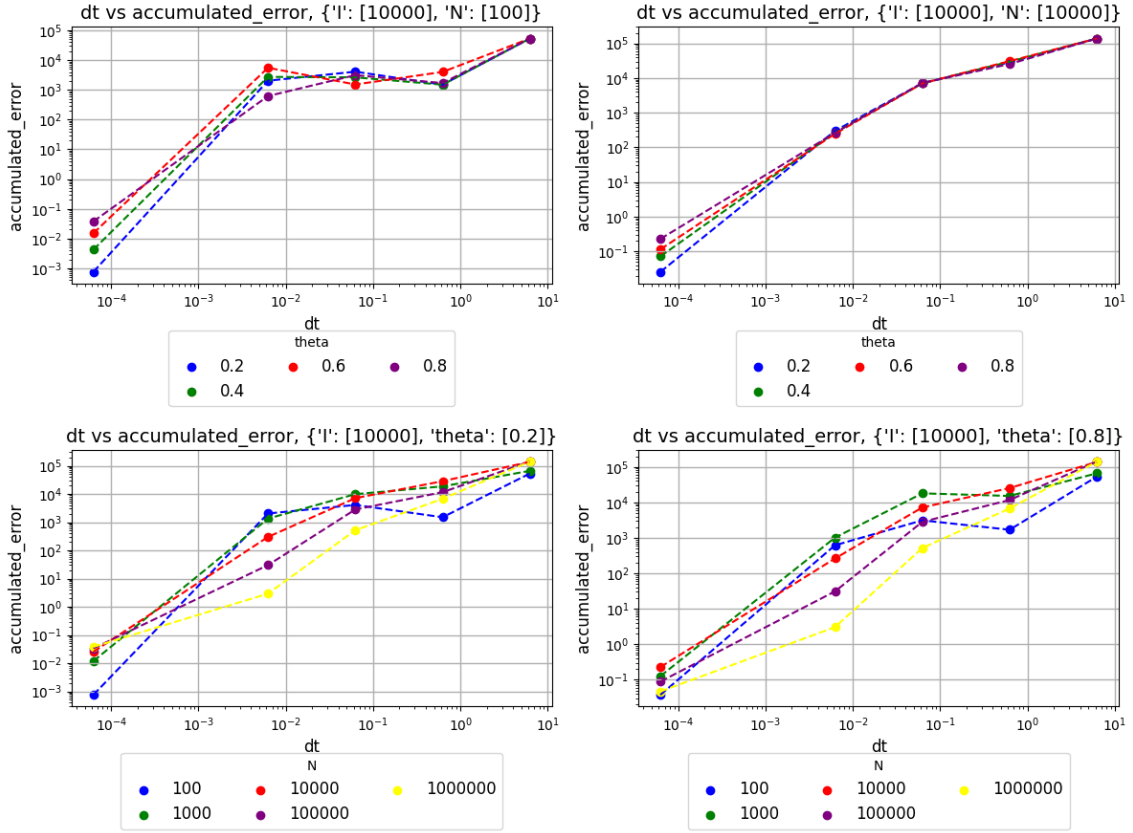


FIGURA 4.8: Curva de  $dt$  vs error acumulado, agrupado por  $\theta$  y  $N$  (escala log)

El error incrementa conforme aumenta  $dt$  en todos los casos evaluados. Para valores pequeños de  $N$  ( $N = 10^2$  y  $N = 10^3$ ), el crecimiento en función de  $dt$  presenta ruido, independientemente del  $\theta$  utilizado. Para valores superiores de  $N$ , el crecimiento es mucho más estable, con las curvas agrupadas por  $\theta$  convergiendo y perdiendo diferenciación a medida que  $dt$  aumenta.

#### 4.1.4.5. Observaciones Finales sobre Características de Simulación

Tras analizar el comportamiento del conjunto de datos, se ha identificado un fuerte nivel de ruido para valores de  $N$  en el rango de  $10^2$ . Pruebas iniciales corroboran que estos introducen comportamientos irregulares el proceso de entrenamiento, afectando la capacidad predictiva de los modelos al tratar de converger sobre un resultado tan irregular. Por lo tanto, se ha decidido limitar el conjunto de datos de entrenamiento a valores de  $N$  superiores a  $10^2$ , buscando un comportamiento mas uniforme en las características.

Adicionalmente, debido a la alta densidad de datos presentes en el parámetro  $I$ , se aplicará un muestreo que seleccionará observaciones cada **1000** iteraciones, buscando mejorar los tiempos de modelado para el entrenamiento y a su vez preservando la representatividad de las tendencias principales en el conjunto de datos.

La combinación de estos dos filtros ha permitido reducir significativamente el tamaño del conjunto de datos, pasando de **1000100** observaciones a un total de **880** filas, manteniendo únicamente las observaciones más relevantes para el entrenamiento de los modelos.

## 4.2 U2. Entrenamiento de Modelos

### 4.2.1 Distribución de características

Se realiza un análisis detallado de la distribución de las características presentes en los conjuntos de datos, tanto para los modelos de ejecución como para los de precisión, con el objetivo de identificar transformaciones necesarias para optimizar el entrenamiento de los modelos.

## Ejecución

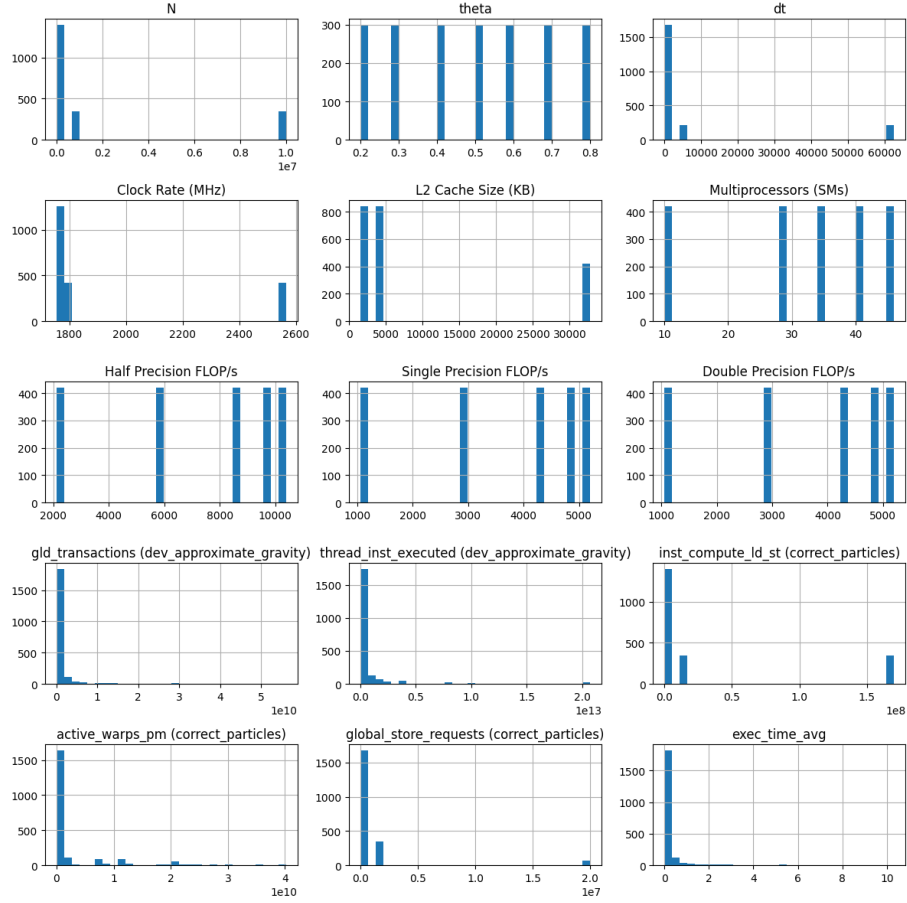


FIGURA 4.9: Características de ejecución sin transformar

La variable  $N$  presenta una distribución exponencial, lo que refleja el rango amplio de valores evaluados durante las simulaciones. De forma similar, la variable  $dt$  sigue un comportamiento exponencial, aunque dentro de un rango más reducido de valores. Cabe destacar que, a partir de pruebas preliminares, se observó una mejora en la precisión del modelo al mantener  $dt$  sin escalado, aplicando únicamente una transformación logarítmica para ajustar su rango.

Por su parte,  $\theta$  exhibe un comportamiento uniforme que abarca un rango de valores comúnmente utilizado en simulaciones. En este caso, únicamente será necesario aplicar un escalado para normalizar sus valores. Algunas características relacionadas con el hardware de las GPUs presentan también tendencias exponenciales; sin embargo, se evaluó que aplicar transformaciones logarítmicas podría desplazar las distribuciones hacia el extremo opuesto, generando un nuevo desequilibrio. Por ello, se optará únicamente por realizar un escalado directo para estas características.

Adicionalmente, las variables de perfilado muestran un comportamiento exponencial más acentuado que  $N$ , lo cual será considerado al aplicar las transformaciones. Por último, la variable objetivo (`exec_time_avg`) presenta una distribución exponencial en un rango reducido, similar al comportamiento de  $dt$ .

## Precisión

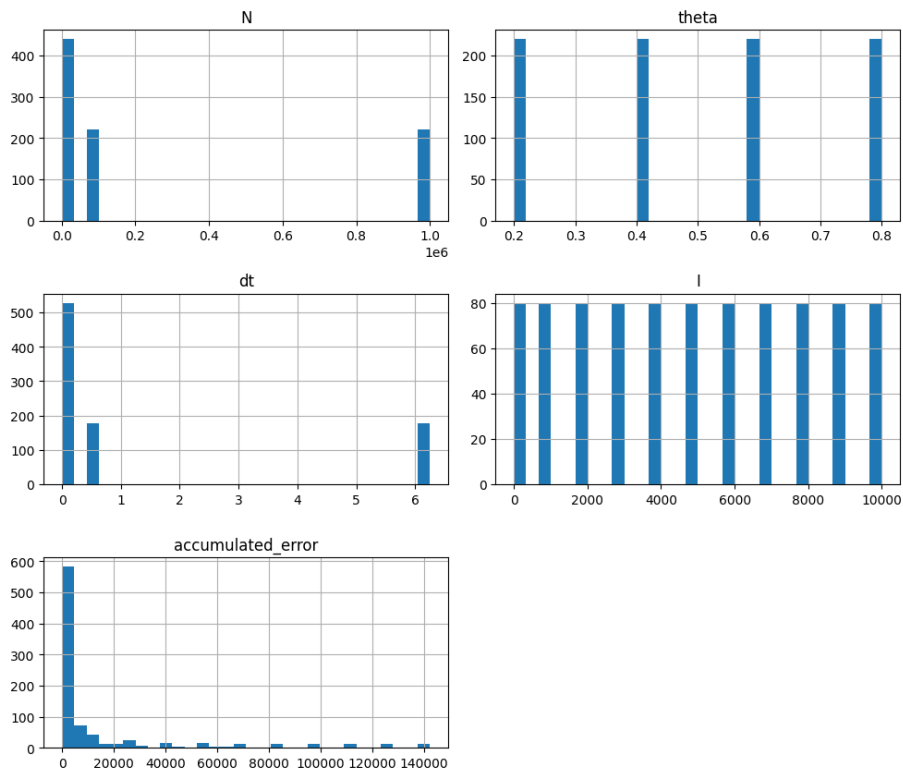


FIGURA 4.10: Características de precisión sin transformar

En el caso del modelo de precisión, las variables  $N$ ,  $\theta$  y  $dt$  mantienen esencialmente las mismas distribuciones observadas en el conjunto de datos de ejecución. La variable  $I$ , en contraste, presenta una distribución uniforme pero con un rango considerablemente amplio. Pruebas preliminares revelaron que aplicar un escalado directo en  $I$  dificultaba la estimación precisa de los valores crecientes de error en función de esta variable.

Finalmente, la variable objetivo del modelo de precisión, correspondiente al error acumulado, también presenta un comportamiento exponencial con un rango reducido, alineándose con el patrón observado en el modelo de ejecución.

#### 4.2.2 Transformaciones por Aplicar

El análisis realizado, junto con las pruebas preliminares de los modelos, ha permitido definir las transformaciones óptimas para cada columna del conjunto de datos.

Transform	Features
StandardScaler	theta, <b>GPU Features</b>
log10Transform	I
logTransform	dt
log10TransformScale	N, <b>Profiling Features</b>
logTransformScale	<b>Target Features</b>

TABLA 4.7: Transformaciones por característica

#### 4.2.3 Distribución Transformada de Características

Se presentan las características transformadas para visualización.

## Ejecución

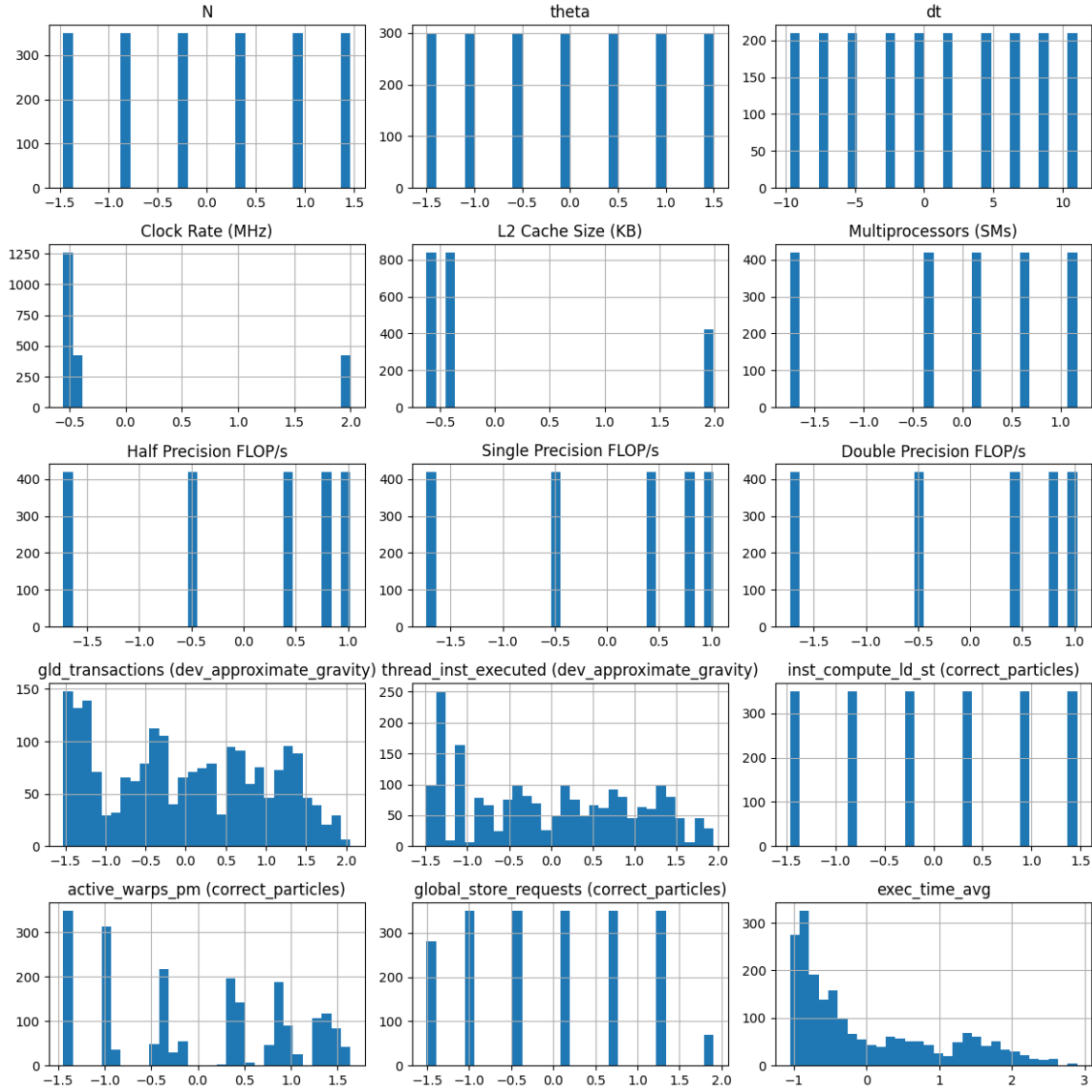


FIGURA 4.11: Características de ejecución transformadas

## Precisión

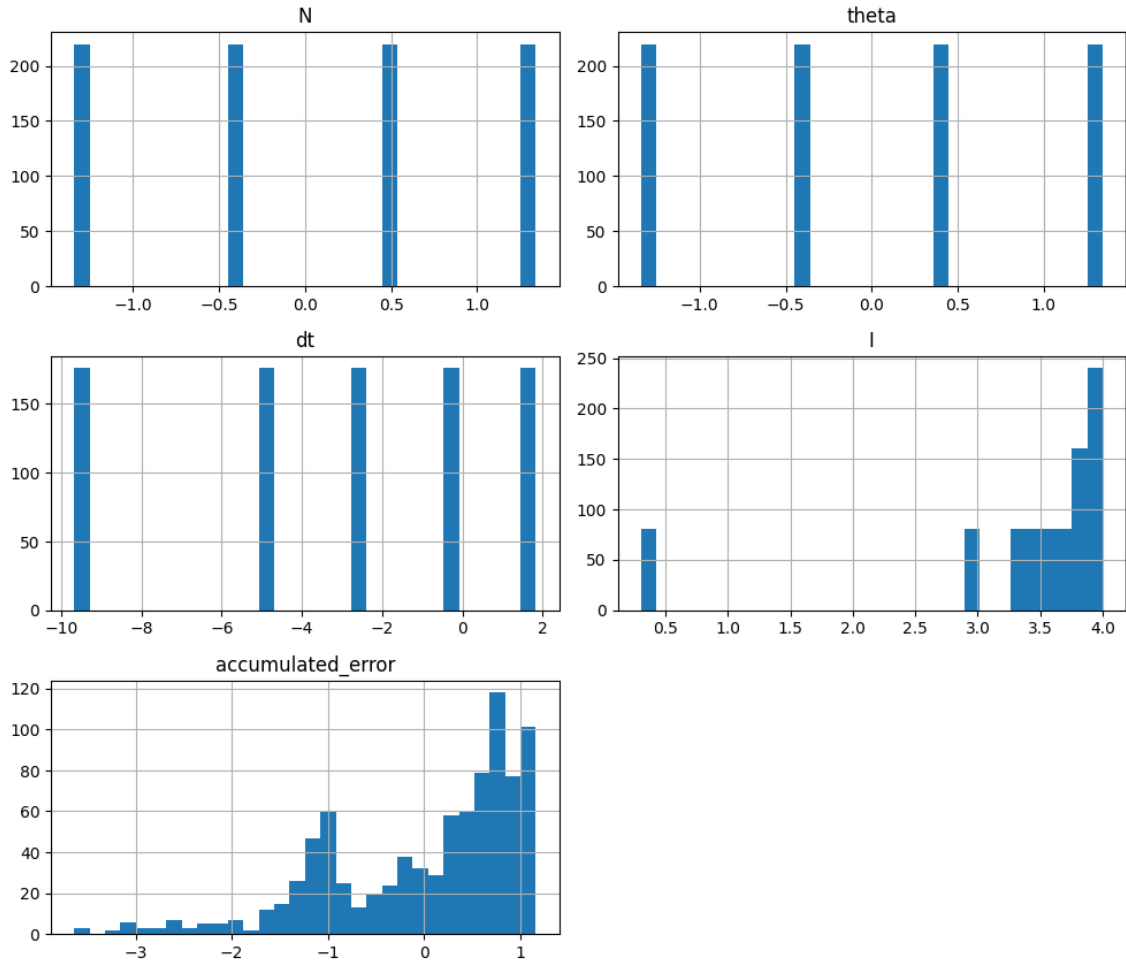


FIGURA 4.12: Características de precisión transformadas

### 4.2.4 Modelado de ejecución

Se presentan los resultados obtenidos tras entrenar y evaluar los modelos de predicción de tiempo de ejecución, utilizando tanto *Random CV* como *GPU CV* como estrategias de *Cross Validation*.



#### 4.2.4.1. Ejecución sin Perfilado

A continuación, se detallan los modelos entrenados para la estimación de tiempos de ejecución sin incluir las características de perfilado:

Regressor	Random CV (10 Fold)		GPU CV (5 GPU)	
	MAPE	MAE	MAPE	MAE
HuberRegressor	66.89 %	0.175	64.64 %	0.177
PolyRidgeRegressor	6.32 %	0.011	33.41 %	0.088
SupportVectorRegresor	9.82 %	0.015	33.56 %	0.106
DecisionTreeRegressor	6.04 %	0.005	19.38 %	0.049
ExtraTreesRegressor	6.26 %	0.008	22.34 %	0.059

TABLA 4.8: Errores finales, utilizando Random CV y GPU CV

Los resultados obtenidos reflejan comportamientos particulares de cada tipo de modelo:

**Modelo Lineal** Este modelo presentó valores altos de MAPE tanto en *Random CV* como en *GPU CV*, evidenciando sus limitaciones para capturar la complejidad de las interacciones entre las características de entrada y la variable objetivo.

**Modelo Polinomial** El modelo polinomial mostró una reducción significativa del error MAPE en *Random CV*, gracias a los grados adicionales que permiten representar mejor las interacciones entre características. Sin embargo, aunque el error también disminuyó en *GPU CV*, esta mejora fue menos notable, indicando limitaciones en la generalización frente a las diferencias en hardware.

**Support Vector Machines (SVM)** El modelo SVM presentó un error MAPE bajo en *Random CV*, lo que demuestra la efectividad de los hiperplanos RBF para modelar el comportamiento de las características. A pesar de ello, el error en *GPU*

*CV* fue comparable al del modelo polinomial, lo que sugiere que las curvas continuas no logran representar completamente las diferencias de hardware.

**Árboles de Decisión** El modelo de Árbol de Decisión directo mostró el error MAPE más bajo en *Random CV*, seguido muy de cerca por el modelo *Extra Trees*. Este comportamiento se atribuye a la capacidad de estos modelos jerárquicos para ajustarse al rango entrenado. En el caso de *GPU CV*, ambos modelos también lograron los valores de MAPE más bajos, de manera contraintuitiva, dado que este tipo de modelos suele carecer de capacidad de interpolación y extrapolación. Esto sugiere que la selección de características durante la búsqueda exhaustiva permitió modelar cierto grado de interpolación en el entrenamiento.

En la tabla siguiente se presentan los errores MAPE detallados por GPU durante la validación *GPU CV*:

GPU	HR		PRR		SVR		DTR		ETR	
	MAPE	MAE	MAPE	MAE	MAPE	MAE	MAPE	MAE	MAPE	MAE
RTX 4060 Ti	54.04 %	0.109	19.39 %	0.023	19.44 %	0.018	17.18 %	0.044	11.81 %	0.005
RTX 3070	66.92 %	0.095	23.71 %	0.026	23.91 %	0.019	14.47 %	0.005	20.72 %	0.029
RTX 3060	90.10 %	0.165	36.25 %	0.058	34.96 %	0.067	24.14 %	0.018	27.76 %	0.047
RTX 2070 SUPER	54.67 %	0.159	22.50 %	0.069	22.86 %	0.065	23.39 %	0.073	21.70 %	0.051
GTX 1060	57.49 %	0.360	65.19 %	0.261	66.62 %	0.362	37.68 %	0.197	37.68 %	0.197

TABLA 4.9: GPU CV, error por GPU

A partir del detalle de errores por GPU, se observa lo siguiente:

**Modelo Lineal** El modelo Lineal presentó errores MAPE consistentemente altos a través de todas las GPUs, con un error particularmente elevado en la RTX 3060.

**Modelo Polinomial** Este mostró el mayor error en la GPU de menores características (GTX 1060) y el menor error en la GPU de mayores características (RTX

4060 Ti). Este comportamiento sugiere una tendencia a favorecer la extrapolación hacia configuraciones de hardware superiores.

**Support Vector Machines (SVM)** El modelo SVM presentó un comportamiento similar al del Polinomial, con limitaciones en la representación de hardware mediante modelos continuos.

**Árboles de Decisión** El modelo de Árbol de Decisión mostró menores errores en la extrapolación para GPUs de menor gama, reduciendo el MAPE en casi la mitad en comparación con modelos continuos. También presentó mejores resultados en la extrapolación hacia hardware superior en comparación con modelos previos.

**Extra Trees** El modelo *Extra Trees* replicó el comportamiento del Árbol de Decisión en la extrapolación hacia hardware inferior, pero logró una reducción aún mayor en la extrapolación hacia hardware superior. No obstante, presentó ligeros incrementos en los errores de interpolación.

#### 4.2.4.2. Ejecución con perfilado

Se presentan los resultados obtenidos para la estimación de tiempos de ejecución utilizando las características de perfilado durante el entrenamiento.

Regressor	Random CV (10 Fold)		GPU CV (5 GPU)	
	MAPE	MAE	MAPE	MAE
HuberRegressor	32.14 %	0.091	1,254.74 %	3.248
PolyRidgeRegressor	5.79 %	0.006	41.33 %	0.144
SupportVectorRegresor	5.56 %	0.007	27.72 %	0.045
DecisionTreeRegressor	6.34 %	0.006	19.69 %	0.058
ExtraTreesRegressor	5.27 %	0.004	17.84 %	0.049

TABLA 4.10: Errores finales con perfilado, utilizando Random CV y GPU CV

Los resultados obtenidos muestran los siguientes comportamientos:

**Modelo Lineal** Al incluir características de perfilado, el error MAPE se redujo a la mitad para *Random CV*, en comparación con el modelo sin perfilado. Sin embargo, en *GPU CV*, el error aumentó considerablemente, limitando la utilidad del modelo para predicciones en diferentes configuraciones de hardware.

**Modelo Polinomial** Este modelo presentó una mejora marginal en *Random CV* (reducción del 0.5 %) frente al modelo sin perfilado. No obstante, el error en *GPU CV* aumentó en un 12 %, lo que sugiere que las características adicionales dificultan la capacidad del modelo para generalizar a configuraciones de hardware distintas.

**Support Vector Machines (SVM)** El modelo SVM logró una reducción del error MAPE de más del 4 % en *Random CV* y de aproximadamente un 5 % en *GPU CV*, lo que evidencia que las características adicionales pueden beneficiar esta categoría de modelos.

**Árboles de Decisión** El modelo de Árbol de Decisión directo mostró un incremento de 0.3 % en el error tanto para *Random CV* como para *GPU CV*, indicando que las características adicionales no aportan mejoras significativas en este tipo de modelos.

**Extra Trees** Este modelo presentó una mejora del 1 % en *Random CV* y de casi el 5 % en *GPU CV*, alcanzando los menores valores de error en ambas métricas. Esto sugiere que las características de perfilado ayudan a los métodos de árboles aleatorios a representar mejor el comportamiento.

En la tabla siguiente se muestran los errores MAPE por GPU durante la validación *GPU CV*:

GPU	HR		PRR		SVR		DTR		ETR	
	MAPE	MAE	MAPE	MAE	MAPE	MAE	MAPE	MAE	MAPE	MAE
RTX 4060 Ti	2,408.11 %	1.199	28.01 %	0.098	22.25 %	0.046	25.81 %	0.047	12.49 %	0.006
RTX 3070	59.87 %	0.053	52.30 %	0.059	43.39 %	0.018	20.70 %	0.038	20.60 %	0.024
RTX 3060	50.22 %	0.154	33.01 %	0.114	18.54 %	0.025	16.76 %	0.039	16.60 %	0.010
RTX 2070 SUPER	37.44 %	0.088	13.35 %	0.036	16.51 %	0.037	26.31 %	0.073	21.56 %	0.052
GTX 1060	3,718.65 %	14.747	79.96 %	0.413	37.91 %	0.099	29.83 %	0.178	19.25 %	0.149

TABLA 4.11: GPU CV, error por GPU

A partir del detalle por GPU, se observa lo siguiente:

**Modelo Lineal** Los errores fueron extremadamente altos en los casos de extrapolación hacia GPUs de menor y mayor gama (GTX 1060 y RTX 4060 Ti). En casos intermedios de interpolación, como con la RTX 3060, los errores mejoraron respecto al modelo sin perfilado.

**Modelo Polinomial** Este modelo mostró un mayor error de extrapolación en GPUs de menor y mayor gama. En los casos de interpolación, las GPUs de menor gama presentaron menores errores, aunque el error se incrementó sustancialmente para la GPU RTX 3070 en comparación con el modelo sin perfilado.

**Support Vector Machines (SVM)** Se observó una mejora en la extrapolación hacia GPUs de menor gama y un empeoramiento hacia GPUs de mayor gama. Similar al modelo polinomial, la interpolación mejoró para GPUs de menor gama, pero empeoró significativamente para GPUs de mayor gama.

**Árboles de Decisión** Este modelo mejoró en la extrapolación hacia GPUs de menor gama, pero empeoró para GPUs de mayor gama. En los casos de interpolación,

los errores aumentaron tanto para GPUs de gama baja como alta, mejorando solo para la RTX 3060.

**Extra Trees** Este modelo mostró una mejora significativa en la extrapolación hacia GPUs de menor gama y un cambio mínimo hacia GPUs de mayor gama. En el resto de GPUs, la interpolación mejoró de manera notable o empeoró de forma mínima, logrando así el menor MAPE promedio entre todos los modelos analizados.

#### 4.2.4.3. Comparación y Selección de Modelos de Ejecución

El modelo **Extra Trees** se establece como el único que presenta una mejora significativa en el error tanto para *Random CV* como para *GPU CV* al incluir las características de perfilado. Sin embargo, no se considera que dicha mejora sea lo suficientemente sustancial como para justificar el proceso adicional de extracción de datos de perfilado.

Si bien se esperaba una mejora notable en los modelos continuos, como el polinomial y *Support Vector Machines*, estas no se observaron en los experimentos realizados. Con base en estos resultados, se establece como modelo final el conjunto de entrenamiento **sin características de perfilado**, recomendando el modelo **Extra Trees** como el más adecuado para la estimación de tiempos de ejecución en GPUs no evaluadas.

Además, es importante destacar que el comportamiento del **Árbol de Decisión directo** en *Random CV* puede no ser completamente indicativo de su bajo error MAPE. Esto se debe a que su naturaleza jerárquica tiende a discretizar los resultados de regresión, lo que puede ser menos útil en escenarios donde se requiere estabilidad y capacidad de interpolación. Por este motivo, se selecciona también a un

modelo continuo como el **Polinomial**, con un error comparable, o el SVR, buscando ofrecer estimaciones más consistentes y estables en *Random CV*.

#### 4.2.5 Modelado de precisión

Se presentan los resultados obtenidos para la estimación del error acumulado, utilizando el conjunto de datos filtrado y muestreado:

Regressor	Random CV (10 Fold)		TS CV (5 Split)	
	MAPE	MAE	MAPE	MAE
HuberRegressor	543.86 %	53,985	313.37 %	128,466
PolyRidgeRegressor	38.50 %	2,051	10.75 %	1,826
SupportVectorRegresor	24.00 %	681	12.46 %	1,078
DecisionTreeRegressor	46.76 %	370	17.55 %	3,031
ExtraTreesRegressor	26.11 %	350	17.55 %	3,031

TABLA 4.12: Errores finales, utilizando Random CV y TimeSeries CV

**Modelo Lineal:** El modelo lineal muestra un error MAPE extremadamente alto tanto para *Random CV* como para *Timeseries CV*, lo que hace inviable su uso para realizar predicciones en este contexto. La simplicidad inherente al modelo lineal no permite capturar la complejidad del comportamiento observado en los datos.

**Modelo Polinomial:** El modelo polinomial presenta una reducción significativa en el error frente al modelo lineal en *Random CV*. Además, logra el menor error en *Timeseries CV*, evidenciando su capacidad para modelar el comportamiento no lineal del error acumulado. Los grados adicionales del polinomio permiten capturar con mayor precisión las dinámicas del sistema, favoreciendo tanto la extrapolación como la interpolación para iteraciones superiores.

**Support Vector Machines (SVM):** Este modelo presenta el menor error MAPE en *Random CV*, lo que indica predicciones más consistentes dentro del rango de parámetros evaluado. En *Timeseries CV*, los errores son los segundos más bajos del conjunto, sugiriendo que las curvas suaves y continuas del modelo SVM permiten un buen desempeño al extrapolar el comportamiento del error acumulado.

**Árbol de Decisión (Directo):** El Árbol de Decisión muestra el segundo error más alto en *Random CV*, reflejando las limitaciones de este enfoque debido a su naturaleza jerárquica y no continua. Los errores en *Timeseries CV* también son de los más altos del conjunto, empatando con el modelo Extra Trees, lo que refuerza su limitada capacidad para extrapolar el comportamiento de iteraciones más allá del rango entrenado.

**Extra Trees:** El modelo Extra Trees presenta un error MAPE en *Random CV* entre los resultados del SVM y el polinomial, lo que indica una mejora significativa frente al Árbol de Decisión base. Este comportamiento sugiere que los conjuntos de árboles aleatorios logran representar mejor la continuidad de las iteraciones en el rango entrenado, aunque los errores en *Timeseries CV* son iguales a los del Árbol de Decisión, mostrando una limitada capacidad para extrapolar.



En la tabla siguiente se presentan los errores MAPE detallados por *timestep* durante la validación de *Timestep CV*:

TS	HR		PRR		SVR		DTR		ETR	
	MAPE	MAE	MAPE	MAE	MAPE	MAE	MAPE	MAE	MAPE	MAE
[6000]	333.54 %	100,789	11.71 %	1,468	17.91 %	818	22.20 %	3,017	22.20 %	3,017
[7000]	317.14 %	114,741	11.63 %	1,631	13.55 %	939	19.55 %	3,025	19.55 %	3,025
[8000]	308.44 %	128,590	10.29 %	1,801	11.38 %	1,102	17.23 %	3,032	17.23 %	3,032
[9000]	304.75 %	142,312	10.14 %	1,995	10.20 %	1,224	15.16 %	3,038	15.16 %	3,038
[10000]	302.97 %	155,898	9.97 %	2,237	9.25 %	1,306	13.61 %	3,044	13.61 %	3,044

TABLA 4.13: TS CV, error por rango de iteraciones

A partir del detalle de errores por *timestep*, se observa lo siguiente:

En términos generales, se observa una reducción del error al aumentar el contexto de los rangos de iteración previos. El modelo polinomial obtiene los errores más bajos desde los primeros *timesteps*, mostrando su efectividad en estimar el comportamiento del error acumulado desde etapas iniciales. Sin embargo, hacia los últimos *timesteps*, el modelo SVM supera al polinomial, logrando el menor error al utilizar todo el contexto previo. Ambos modelos basados en árboles (Árbol de Decisión y Extra Trees) presentan un comportamiento idéntico durante la evaluación de extrapolación, lo que destaca sus limitaciones jerárquicas para este tipo de tarea.

#### 4.2.5.1. Comparación y Selección de Modelos de Precisión

En base a los resultados presentados, se recomendaría utilizar el modelo **Support Vector Regression** (SVR) para estimaciones dentro del rango entrenado, debido a su estabilidad y bajo error MAPE en *Random CV*. Este modelo destaca por su capacidad de representar adecuadamente el comportamiento en configuraciones conocidas. También se recomendaría utilizar el modelo **Extra Trees**, en caso se requiera una mayor precisión a costo de una reducida capacidad de interpolación.

Por otro lado, se recomendaría el modelo **Polinomial** para la extrapolación de errores acumulados a iteraciones superiores ( $I$ ), al demostrar una capacidad superior de modelado de relaciones no lineales. Este modelo presenta el menor error MAPE en validación *Timestep CV*, mostrando su efectividad en escenarios de extrapolación.

### 4.3 U3. Visualización y Estimación

#### 4.3.1 Software de Visualización

El software implementado se encuentra disponible en el repositorio principal de este trabajo. Puede ser ejecutado localmente siguiendo la documentación proporcionada en el mismo repositorio, la cual detalla los pasos necesarios para clonarlo y configurarlo en un entorno local. Además, se tiene la intención de publicar la aplicación web implementada en línea, y el enlace correspondiente será incluido en la documentación para facilitar su acceso.

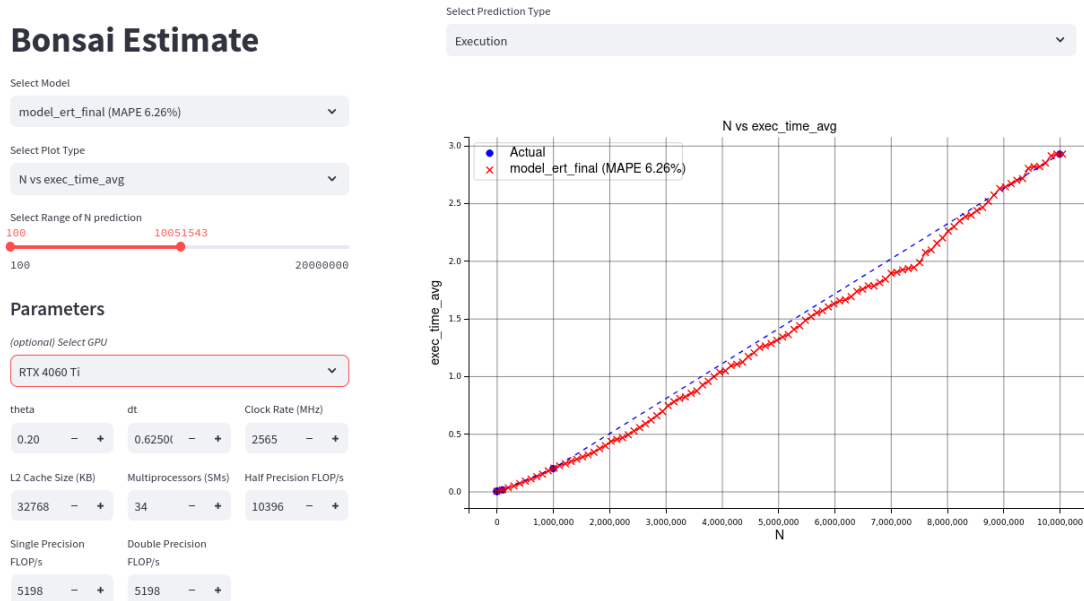


FIGURA 4.13: Implementación Final

### 4.3.2 Visualización y Análisis de Modelos

En esta sección, se presentan ejemplos de predicción obtenidos a partir de los modelos entrenados, utilizando el software de visualización implementado. Se evalúa el comportamiento de los parámetros de simulación en función de los modelos utilizados.

#### 4.3.2.1. Ejecución

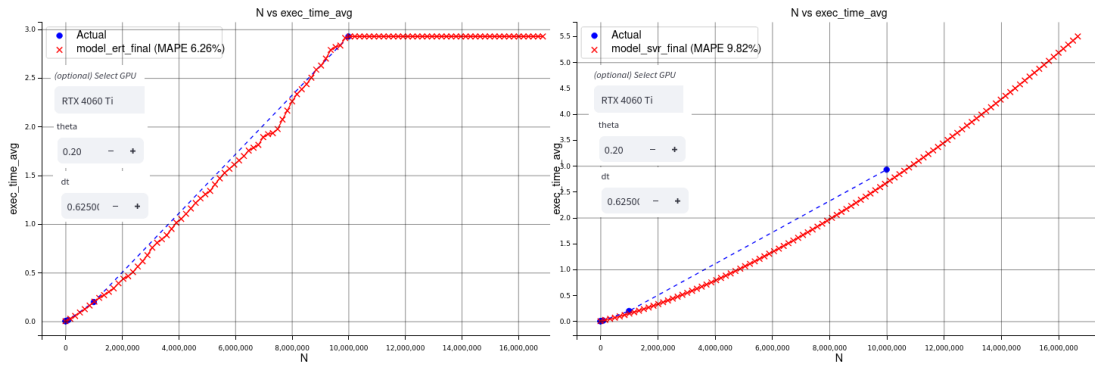


FIGURA 4.14: Predicción y extrapolación N

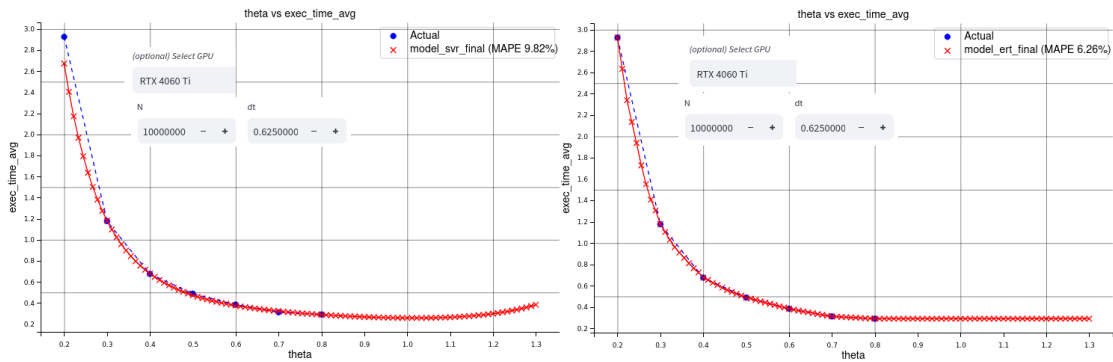


FIGURA 4.15: Predicción y extrapolación theta

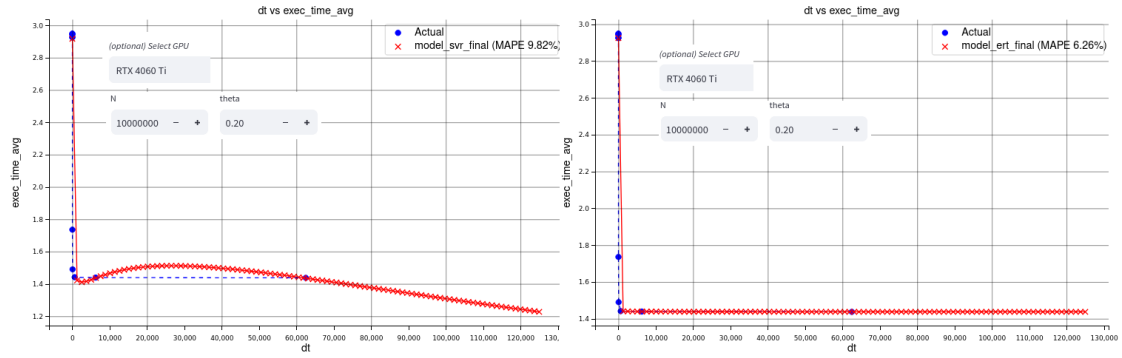


FIGURA 4.16: Predicción y extrapolación dt

Los resultados presentados corroboran lo indicado en los análisis previos, donde se observa que el modelo basado en árboles *ExtraTrees* presenta dificultades al extrapolar datos más allá del rango observado durante el entrenamiento. No obstante, este modelo exhibe una precisión notablemente mayor cuando se consulta sobre datos dentro del rango de características entrenadas. Por otro lado, el modelo *Support Vector Regression (SVR)* demuestra una mayor capacidad de extrapolación hacia valores no observados durante el entrenamiento, aunque a costa de una menor precisión en el ajuste de las curvas a los datos reales.

#### 4.3.2.2. Precisión

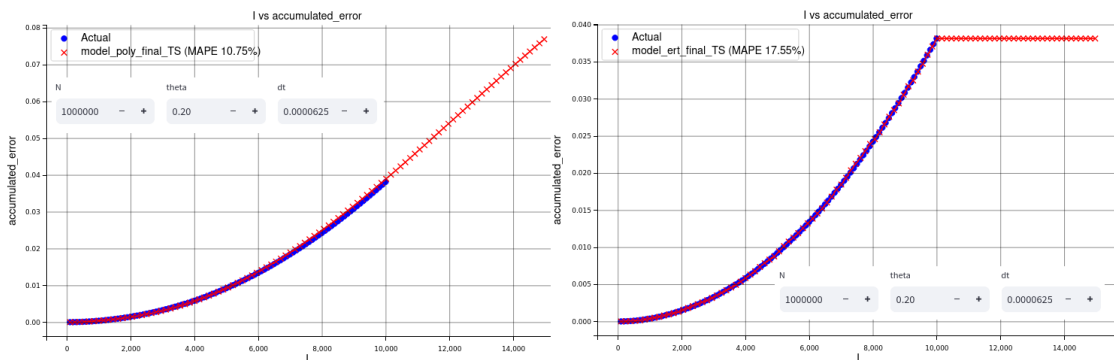


FIGURA 4.17: Predicción y extrapolación I

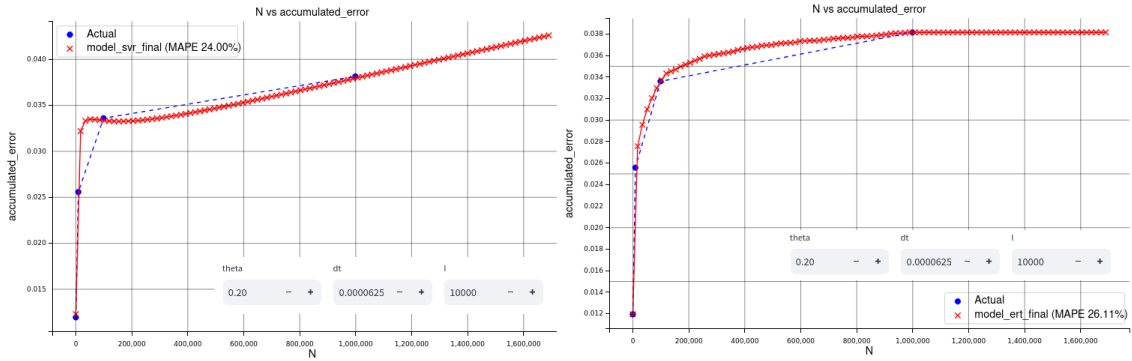


FIGURA 4.18: Predicción y extrapolación N

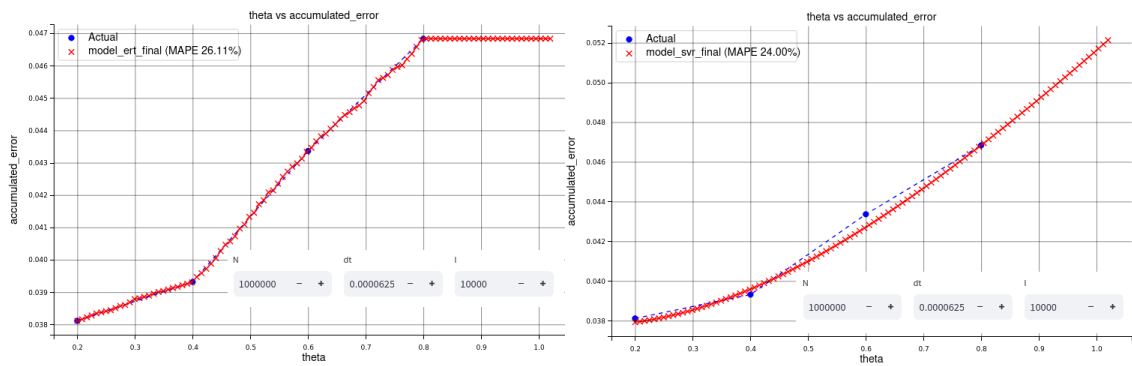


FIGURA 4.19: Predicción y extrapolación theta

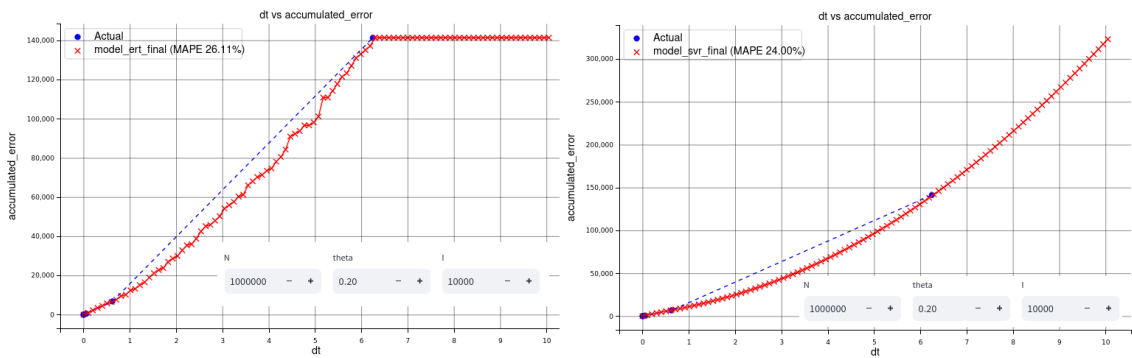


FIGURA 4.20: Predicción y extrapolación dt

Los resultados de precisión reflejan un comportamiento similar al observado en la ejecución de los modelos. Los modelos basados en árboles presentan una alta precisión dentro del rango esperado, aunque muestran dificultades al extrapolar

fuera de dicho rango. En contraste, los modelos *SVR* y *polinomiales* tienen un mejor desempeño al extrapolar fuera del rango observado, aunque esto conlleva una reducción en la precisión dentro del rango de entrenamiento. Este comportamiento se acentúa especialmente para la variable principal de este modelo, que corresponde al rango de iteración  $I$ . Las curvas mostradas fueron generadas utilizando los modelos entrenados mediante *Timeseries CV*, destacando una notable capacidad de extrapolación respecto a esta variable.

#### **4.3.3 Evaluación de Comportamiento de GPUs Fuera de Alcance**

En esta sección, se evalúa el comportamiento de las GPUs fuera del rango de entrenamiento, con el objetivo de visualizar la capacidad de interpolación y extrapolación de los modelos de ejecución.

#### 4.3.3.1. Tesla T4

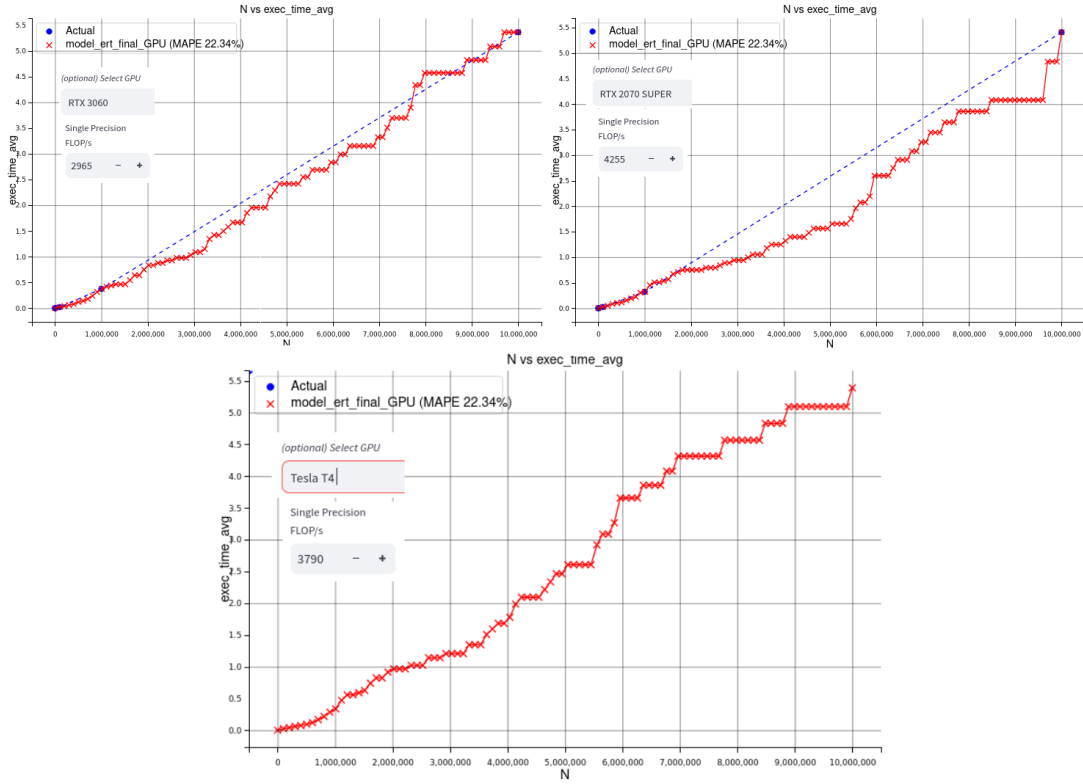


FIGURA 4.21: Graficas generadas para gpus semejantes

En esta evaluación, se utiliza el modelo Extra Trees con GPU CV, el cual emplea únicamente los FLOPs como característica representativa de las GPUs. Se generan datos para las GPUs RTX 3060 y RTX 2070 SUPER, ya que la Tesla T4 se encuentra en un punto intermedio en cuanto a cantidad de FLOPs, lo que representa un caso de interpolación.

Al generar la estimación para la Tesla T4, se observa un comportamiento acorde con lo esperado. Los tiempos de ejecución para la RTX 3060 y RTX 2070 SUPER son casi idénticos ( 5.25s para  $N = 10^7$ ), y las estimaciones generadas para la

Tesla T4 coinciden con estos valores, lo que valida la predicción del comportamiento de esta GPU no vista previamente.

#### 4.3.3.2. Tesla P100

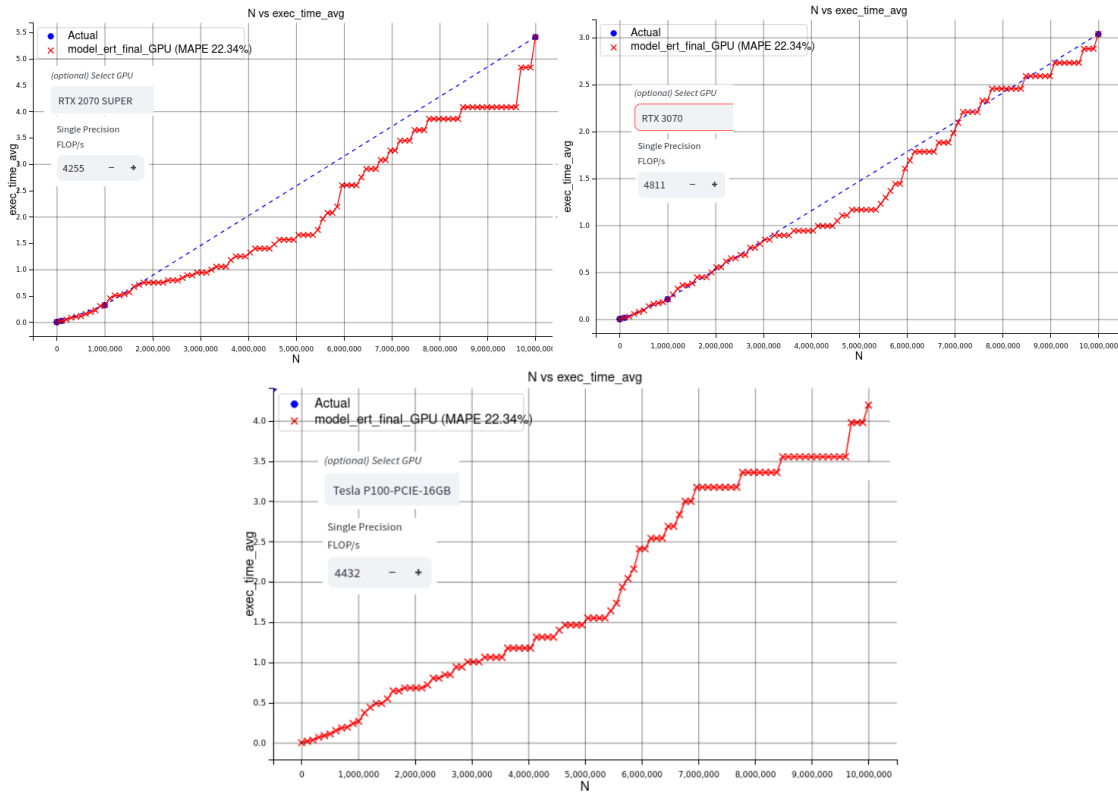


FIGURA 4.22: Graficas generadas para gpus semejantes

Se realiza una nueva evaluación utilizando el modelo Extra Trees con GPU CV. En este caso, se generan datos para la RTX 2070 SUPER y la RTX 3070 en base a los FLOPs de la Tesla P100, lo que nuevamente presenta un caso de interpolación.

Este rango de FLOPs muestra un cambio significativo en los tiempos de ejecución, pasando de 5.25s a 3s para  $N = 10^7$ . La estimación para la P100 arroja un tiempo de 4.25s para el mismo valor de  $N$ , lo cual concuerda con el



comportamiento esperado, considerando la diferencia en los FLOPs entre cada GPU: 2070 (4255), P100 (4432), y 3070 (4811).

#### 4.3.3.3. A100

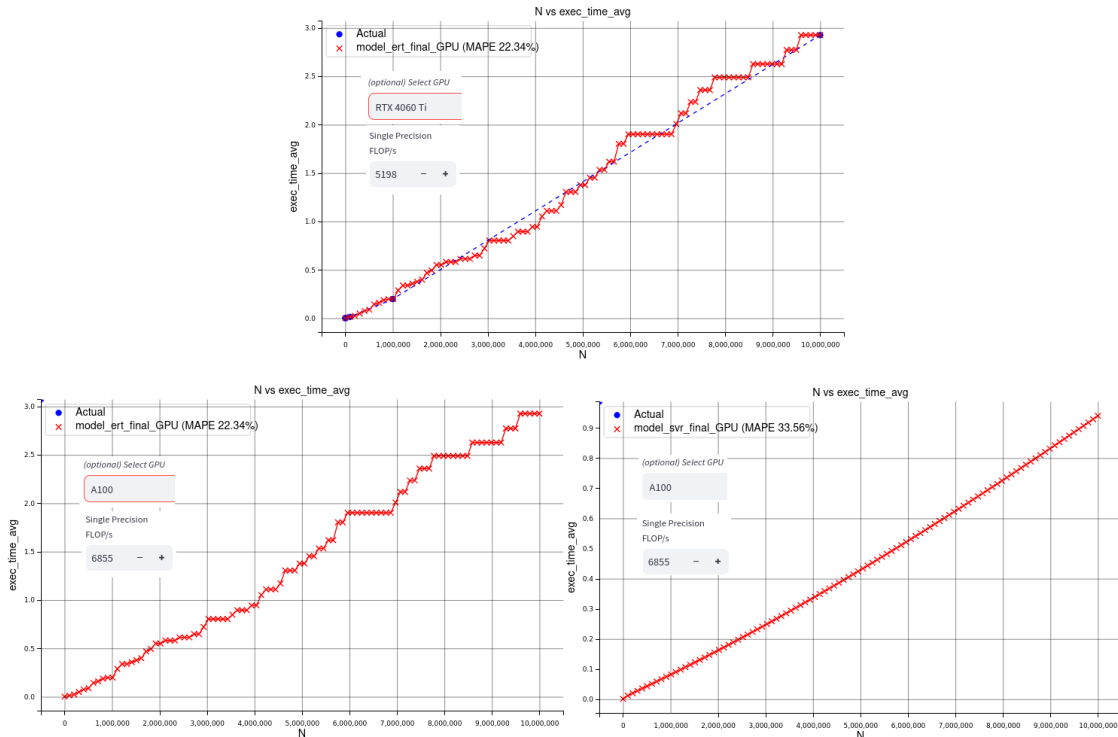


FIGURA 4.23: Grafica de gpu semejante y estimaciones bajo dos modelos distintos

Inicialmente, se realiza la evaluación utilizando el modelo Extra Trees con GPU CV. Se generan datos para la RTX 4060 Ti, una GPU con mayores FLOPs dentro del conjunto de entrenamiento, que presenta más de 1600 FLOPs de diferencia respecto a la A100. Este escenario representa un caso significativo de extrapolación.

Al realizar la evaluación, se observa que el modelo Extra Trees no logra extrapolar correctamente para un valor de FLOPs superior al de la RTX 4060 Ti, lo cual se explica por el comportamiento jerárquico de este modelo. Para mitigar esta limitación, se realiza una nueva evaluación utilizando el modelo SVR con GPU CV, que

logra extrapolar un tiempo de ejecución de aproximadamente 0.9s para  $N = 10^7$ , lo cual tiene sentido considerando la diferencia significativa en FLOPs entre la mejor GPU entrenada y la A100.

#### 4.3.3.4. Evaluación de Comportamiento de *Selene*

Con el objetivo de extrapolar el comportamiento de nuestros modelos a un contexto de HPC, se intenta verificar cómo se comportan los clusters de GPUs A100 utilizados en Selene.

Cada uno de los clusters de Selene está compuesto por 8 GPUs A100, por lo que se realiza un escalado inicial a esta configuración de hardware.

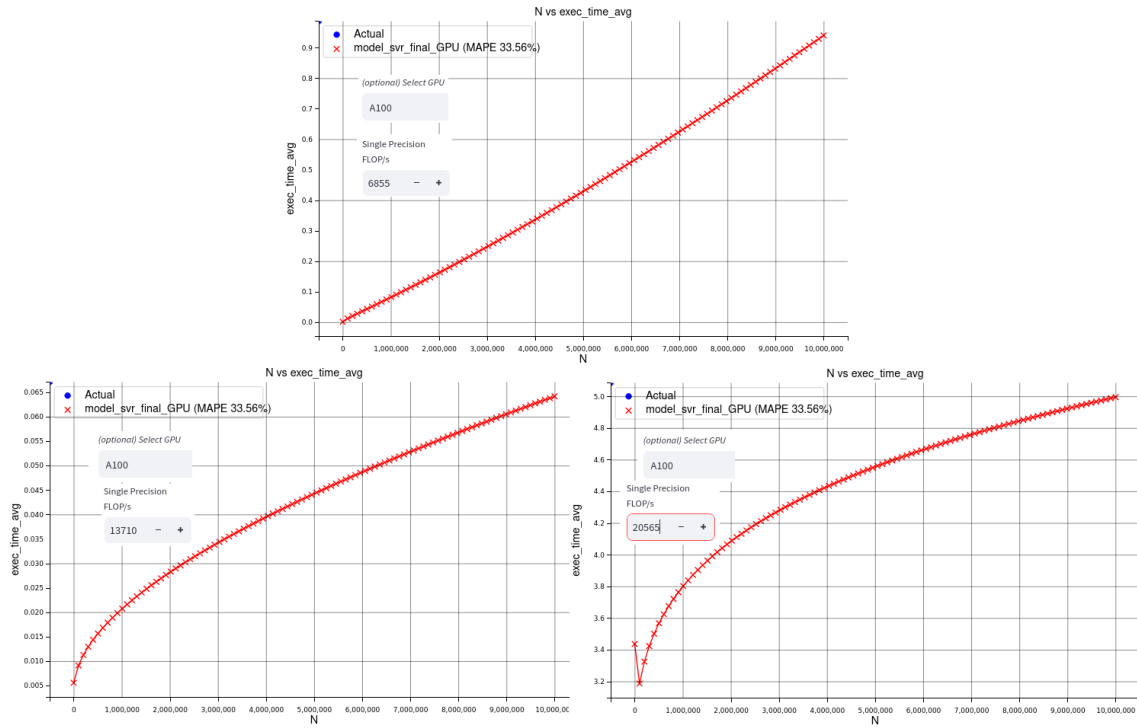


FIGURA 4.24: Grafica de gpu semejante y estimaciones bajo dos modelos distintos

Las estimaciones generadas abarcan configuraciones de A100x1, A100x2 y A100x3. Al verificar el comportamiento, se observa que el modelo SVR con GPU CV

logra representar adecuadamente el comportamiento de las GPUs hasta un cluster de 2 GPUs (13710 FLOPs). Sin embargo, por encima de este punto, el modelo falla, generando gráficas incoherentes que no reflejan un comportamiento esperado, tanto para el modelo SVR como para cualquier otro modelo disponible.

Debido a esto, no es posible realizar una estimación confiable del comportamiento de un solo cluster de Selene, y mucho menos de los 540 clusters que conforman el supercomputador. La diferencia en capacidad computacional es demasiado grande para que nuestros modelos puedan extrapolar a tanta distancia de los datos de prueba disponibles.

## CONCLUSIONES

Este trabajo logró establecer un proceso robusto de perfilado y modelado del comportamiento de la simulación *Bonsai* en GPUs. Este proceso permitió obtener datasets completos y consistentes para el entrenamiento de modelos, dejando además una metodología adaptable para iteraciones futuras, que puede beneficiarse de la incorporación de datos adicionales provenientes de configuraciones de hardware más diversas, aumentando la robustez y generalización de los modelos.

El filtrado y preprocesamiento de características se validó satisfactoriamente, logrando un conjunto reducido pero representativo de variables clave que sustentaron el entrenamiento de los modelos. Asimismo, los procesos de transformación y el flujo de entrenamiento fueron validados al obtener resultados consistentes y alineados con los objetivos planteados.

Una de las observaciones clave es que las métricas de perfilado utilizadas no mejoraron significativamente el desempeño de los modelos. Esto contrasta con trabajos relacionados, posiblemente debido a la naturaleza multi-kernel de *Bonsai* y a las diferencias en los objetivos de modelado. Mientras que otros estudios buscan predecir comportamientos para kernels pequeños y estables provenientes de conjuntos de *benchmarks*, este trabajo se enfocó en un sistema complejo con dinámicas menos predecibles.

Se identificaron los mejores modelos según las tareas específicas:

### Modelos de ejecución:

- **Extra Trees** mostró el mejor desempeño para consultas dentro del rango de características entrenadas (MAPE 6.26 % para Random CV y 22.34 % para GPU CV).

- **SVR** demostró ser más efectivo en tareas de extrapolación, especialmente para GPUs no entrenadas, aunque con menor precisión en datos dentro del rango (MAPE 9.82 % para Random CV y 33.56 % para GPU CV).

### Modelos de precisión:

- **Extra Trees** resultó más preciso para consultas dentro del rango (MAPE 26.11 % con Random CV y 17.55 % con Timeseries CV). - El modelo **polinomial** destacó en la extrapolación de iteraciones  $I$  (MAPE 38.50 % con Random CV y 10.75 % con Timeseries CV). - **SVR** fue el mejor para extrapolar características individuales restantes (MAPE 24 % con Random CV y 12.46 % con Timeseries CV).

Los resultados obtenidos están en línea con trabajos relacionados, coincidiendo con errores MAPE de trabajos similares, validando la metodología y extendiendo su aplicabilidad desde un único kernel hasta un sistema multi-kernel. Además, se implementó con éxito un software de visualización que facilita la exploración de los modelos candidatos generados en la fase de entrenamiento.

En cuanto a la capacidad de extrapolación, se observó que:

1. **Extra Trees** presenta una notable precisión dentro del rango entrenado, pero tiene limitaciones significativas al extrapolar.

2. **SVR** mostró un desempeño superior para extrapolaciones extremas, aunque con menor precisión dentro del rango entrenado.

3. El modelo **polinomial**, combinado con una validación tipo Timeseries CV, demostró una capacidad destacable de extrapolación, especialmente para la variable de iteraciones  $I$ .

Estos permitieron interpolar y extrapolar resultados de ejecución para un conjunto separado de GPUs no evaluadas, en base a solamente sus características de hardware. Sin embargo, al intentar extrapolar para GPUs con características significativamente superiores a las entrenadas, como los clusters del supercomputador *Selene* (540 clusters de 8 GPUs A100), nuestros modelos no lograron producir estimaciones confiables para una escala mayor a 13710 FLOPs. Este límite de extrapolación se debe a la gran diferencia de capacidad computacional ( 8500 FLOPs más) respecto a la GPU más potente del conjunto de entrenamiento, lo que subraya la necesidad de datos adicionales para extender el alcance del modelo.

Finalmente, este trabajo sienta las bases para futuras investigaciones en la predicción de simulaciones multi-kernel, destacando la importancia de expandir el rango de entrenamiento y explorar nuevas arquitecturas de modelado que puedan abordar de manera más efectiva los desafíos de extrapolación en entornos de HPC.

## RECOMENDACIONES

Con base en los hallazgos y limitaciones identificadas en este trabajo, se plantean las siguientes recomendaciones para futuras investigaciones:

1. **Descartar el uso de métricas de perfilado de kernels:** Se observó que estas métricas no contribuyen significativamente a mejorar el desempeño de los modelos. Como alternativa, se sugiere utilizar herramientas de perfilado para obtener directamente los tiempos de ejecución de los kernels más importantes del sistema. Estos tiempos pueden ser modelados de manera independiente, permitiendo construir modelos más precisos y específicos que eviten la dependencia del tiempo de ejecución total calculado por la simulación.

2. **Exploración de rangos más detallados para los parámetros de entrada:** En este trabajo, se emplearon valores de  $N$  en potencias de 10. Para mejorar la granularidad de los modelos, se recomienda probar valores en potencias de 2 o emplear técnicas de muestreo más finas, similares a las aplicadas para la variable de iteraciones  $I$  en el modelo de precisión. Esto permitiría un análisis más detallado y la implementación efectiva de validación cruzada agrupada por series temporales (*Time Series Cross Validation*).

3. **Generalización para otras distribuciones de partículas:** Si se busca extender la aplicabilidad del modelo a distribuciones de partículas diferentes, sería beneficioso implementar un preanálisis que permita caracterizar la distribución a

predecir. Este análisis podría ser utilizado como un nuevo parámetro de entrada en los modelos, mejorando su capacidad para adaptarse a dinámicas de simulación más diversas.

**4. Desarrollo de modelos híbridos o conjuntos:** Una posible mejora sería combinar modelos polinomiales o de SVR con modelos jerárquicos como Extra Trees. Este enfoque tiene el potencial de aprovechar la precisión de los modelos jerárquicos dentro del rango de entrenamiento, junto con la capacidad de interpolación y extrapolación de los modelos polinomiales. El uso de modelos híbridos podría proporcionar un equilibrio entre precisión y flexibilidad en la predicción.

**5. Ampliación del rango de GPUs y hardware:** Dada la limitada capacidad de extrapolación observada en este trabajo, se recomienda expandir el conjunto de datos de entrenamiento con GPUs de mayor capacidad, que cubran un rango más amplio de características computacionales. Esto permitirá a los modelos manejar mejor casos extremos y entornos de HPC como los clusters de Selene.

En base a estas recomendaciones, se espera que trabajos a futuro logren mejorar la precisión, robustez y generalización de los modelos, sentando las bases para la predicción de simulaciones complejas en entornos de computación avanzada.



## REFERENCIAS BIBLIOGRÁFICAS

- [1] P. Assiroj, A. Hananto, A. Fauzi, and H. L. H. S. Warnars, “High Performance Computing (HPC) Implementation: A Survey,” in *2018 International Conference on Information and Advanced Processing (INAPR)*, 2018, pp. 213–217.
- [2] A. Brandt, “On distributed gravitational n-body simulations,” 2022. [Online]. Available: <https://arxiv.org/abs/2203.08966>
- [3] M. Amaris, R. Camargo, D. Cordeiro, A. Goldman, and D. Trystram, “Evaluating execution time predictions on gpu kernels using an analytical model and machine learning techniques,” *Journal of Parallel and Distributed Computing*, vol. 171, pp. 66–78, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0743731522001903>
- [4] O. Olsson, “Constructing high-quality bounding volume hierarchies for n-body computation using the acceptance volume heuristic,” *Astronomy and Computing*, vol. 22, pp. 1–8, 2018. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2213133717301129>
- [5] J. Bédorf, E. Gaburov, and S. Portegies Zwart, “A sparse octree gravitational n-body code that runs entirely on the gpu processor,” *Journal of Computational Physics*, vol. 231, no. 7, pp. 2825–2839, 2012. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0021999111007364>

- [6] B. Lange and P. Fortin, “Parallel dual tree traversal on multi-core and many-core architectures for astrophysical n-body simulations,” in *Euro-Par 2014 Parallel Processing*, F. Silva, I. Dutra, and V. Santos Costa, Eds. Cham: Springer International Publishing, 2014, pp. 716–727.
- [7] E. Gaburov, S. Harfst, and S. P. Zwart, “Sapporo: A way to turn your graphics cards into a grape-6,” *New Astronomy*, vol. 14, no. 7, pp. 630–637, 2009. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1384107609000359>
- [8] K. Kofler, D. Steinhauser, B. Cosenza, I. Grasso, S. Schindler, and T. Fahringer, “Kd-tree based n-body simulations with volume-mass heuristic on the gpu,” in *Proceedings of the International Parallel and Distributed Processing Symposium, IPDPS*, 11 2014.
- [9] H. C. Plummer, “On the Problem of Distribution in Globular Star Clusters: (Plate 8.),” *Monthly Notices of the Royal Astronomical Society*, vol. 71, no. 5, pp. 460–470, 03 1911. [Online]. Available: <https://doi.org/10.1093/mnras/71.5.460>
- [10] S. Portegies Zwart, S. McMillan, D. Groen, A. Gualandris, M. Sipior, and W. Vermin, “A parallel gravitational n-body kernel,” *New Astronomy*, vol. 13, no. 5, pp. 285–295, 2008. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1384107607001194>
- [11] L. Braun, S. Nikas, C. Song, V. Heuveline, and H. Fröning, “A simple model for portable and fast prediction of execution time and power consumption of gpu kernels,” *ACM Trans. Archit. Code Optim.*, vol. 18, no. 1, Dec. 2021. [Online]. Available: <https://doi.org/10.1145/3431731>
- [12] L. V. Lanker, H. Taboada, E. Brunet, and F. Trahay, “Predicting GPU kernel’s performance on upcoming architectures,” in *The 30th International European*

- Conference on Parallel and Distributed Computing (Euro-Par)*, Madrid, Spain, Aug. 2024. [Online]. Available: <https://hal.science/hal-04614350>
- [13] M. Burtscher and K. Pingali, *An Efficient CUDA Implementation of the Tree-Based Barnes Hut N-Body Algorithm*, 2011.
  - [14] D. P. Mehta and S. Sahni, *Handbook of Data Structures and Applications, Second Edition*, 2nd ed. Chapman & Hall/CRC, 2018, ch. 20: Quadrees and Octrees.
  - [15] NVIDIA Corporation, *CUDA Profiler User's Guide*, 2024, release 12.6, September 24, 2024. [Online]. Available: [https://docs.nvidia.com/cuda/pdf/CUDA\\_Profiler\\_Users\\_Guide.pdf](https://docs.nvidia.com/cuda/pdf/CUDA_Profiler_Users_Guide.pdf)
  - [16] *Nsight Compute Documentation*, NVIDIA, 2024, latest, for the CUDA Toolkit 12.6 Update 2 release and docs, September 30, 2024. [Online]. Available: <https://docs.nvidia.com/nsight-compute/2024.3/>
  - [17] “Univariate feature selection,” [https://scikit-learn.org/stable/modules/feature\\_selection.html#univariate-feature-selection](https://scikit-learn.org/stable/modules/feature_selection.html#univariate-feature-selection), n.d., accessed: 2024-12-01.
  - [18] “Linear models,” [https://scikit-learn.org/stable/modules/linear\\_model.html](https://scikit-learn.org/stable/modules/linear_model.html), n.d., accessed: 2024-12-01.
  - [19] “Svm regression,” <https://scikit-learn.org/stable/modules/svm.html#svm-regression>, n.d., accessed: 2024-12-01.
  - [20] “Decision trees,” <https://scikit-learn.org/stable/modules/tree.html#tree>, n.d., accessed: 2024-12-01.
  - [21] “Random forests and other randomized tree ensembles,” <https://scikit-learn.org/stable/modules/ensemble.html#random-forests-and-other-randomized-tree-ensembles>, n.d., accessed: 2024-12-01.

- [22] “Model evaluation: quantifying the quality of predictions,” [https://scikit-learn.org/stable/modules/model\\_evaluation.html](https://scikit-learn.org/stable/modules/model_evaluation.html), n.d., accessed: 2024-12-01.
- [23] “Cross-validation: evaluating estimator performance,” [https://scikit-learn.org/stable/modules/cross\\_validation.html](https://scikit-learn.org/stable/modules/cross_validation.html), n.d., accessed: 2024-12-01.
- [24] “Tuning the hyper-parameters of an estimator,” [https://scikit-learn.org/stable/modules/grid\\_search.html#tuning-the-hyper-parameters-of-an-estimator](https://scikit-learn.org/stable/modules/grid_search.html#tuning-the-hyper-parameters-of-an-estimator), n.d., accessed: 2024-12-01.
- [25] NVIDIA, “Understanding selene: A modular nvidia supercomputer: Gtc digital april 2021: Nvidia on-demand,” Apr 2021. [Online]. Available: <https://www.nvidia.com/en-us/on-demand/session/gtcspring21-s31700/>

## ANEXOS

Todo el procedimiento de generación de datasets, entrenamiento y software de visualización se encuentran almacenados en el siguiente repositorio:

<https://github.com/Mauricio-Bernuy/Bonsai-estimate>

Todas las gráficas y tablas presentes en este documento, excepto las mencionadas explícitamente, son de elaboración propia.