Universidad Autónoma de Querétaro

Facultad de Ingeniería

Ingeniería Física

Uso de algoritmos de redes neuronales para la resolución de problemas numéricos en física

disminuyendo el costo computacional

Tesis

Que presenta:

Abel Santillán Rodríguez

Director:

Dr. Marco Antonio Aceves Fernández

Codirector:

Dr. Octavio Valenzuela Tijerino

Santiago de Querétaro, Querétaro, México

10 de marzo de 2026

Universidad Autónoma de Querétaro
Facultad de Ingeniería
Licenciatura en Ingeniería Física

Uso de algoritmos de redes neuronales para la resolución de problemas numéricos en física, disminuyendo el costo computacional

TESIS

Que como parte de los requisitos para obtener el grado de
INGENIERO FÍSICO

Presenta:
ABEL SANTILLAN RODRIGUEZ

Dirigido por:
Dr. Marco Antonio Aceves Fernández

Co-dirigido por:
Dr. Octavio Valenzuela Tijerino

Dr. Marco Antonio Aceves Fernández	_____
Presidente	Firma
Dr. Octavio Valenzuela Tijerino	_____
Secretario	Firma
Dr. Josúe de Jesús Trejo Alonso	_____
Sinodal	Firma
Dr. Alberto Hernandez Almada	_____
Sinodal	Firma

Centro Universitario, Santiago de Querétaro, Qro.
Fecha de aprobación por el H. Consejo Universitario [mes y año]
México

Resumen

Esta tesis presenta un estudio comprehensivo del Operador Neuronal de Fourier (FNO) como una alternativa para resolver ecuaciones diferenciales parciales (EDP) en física computacional, con énfasis particular en reducir los costos computacionales comparados con métodos numéricos tradicionales.

La creciente demanda de herramientas de simulación eficientes en campos como la dinámica de fluidos, transferencia de calor y física de plasmas ha motivado la exploración de enfoques basados en aprendizaje automático. Los métodos numéricos tradicionales, aunque robustos y bien establecidos, enfrentan limitaciones computacionales significativas al tratar con simulaciones de alta resolución o exploración extensiva de espacios de parámetros. Este trabajo investiga si los operadores neuronales pueden superar estas limitaciones manteniendo niveles de precisión aceptables.

La metodología involucró implementar la arquitectura FNO usando PyTorch y entrenarla en conjuntos de datos generados a partir de soluciones numéricas espectrales de las ecuaciones de Navier-Stokes 2D y de calor. El modelo fue entrenado con 1,200 muestras con resolución espacial de 64×64, usando una configuración de 6 modos de Fourier, 20 canales internos y 4 capas de Fourier. Se realizó una campaña experimental sistemática para caracterizar el efecto de hiperparámetros clave: modos de Fourier (2-32), ancho de red (10-40 canales) y profundidad de red (2-32 capas).

Los resultados demuestran que el FNO alcanza un error L2 relativo del 2.1 % en el conjunto de validación, significativamente por debajo del umbral del 10 % establecido en la hipótesis de investigación. El modelo captura exitosamente estructuras coherentes del flujo y dinámica de vorticidad en regímenes turbulentos. Además, la red exhibe capacidades de super-resolución notables, generalizando desde la resolución de entrenamiento 64×64 hasta 256×256 con solo un incremento marginal en el error (de 2.1 % a 2.9 %).

El análisis computacional revela que el tiempo de inferencia escala linealmente con el ancho de red, aumentando en un factor de 3.16 al pasar de 10 a 40 canales. El tiempo de entrenamiento exhibe escalamiento aproximadamente lineal con el número de capas, siguiendo la relación $t_{epoch} \approx 2.2 \times n_{capas} + 3.5$ segundos. Estos hallazgos proporcionan insights valiosos para optimizar arquitecturas FNO en aplicaciones prácticas.

La principal limitación identificada es el requerimiento de datos de entrenamiento generados mediante métodos numéricos tradicionales, lo cual representa una inversión computacional que debe amortizarse sobre múltiples inferencias. Sin embargo, una vez entrenado, el FNO permite predicción rápida de campos de solución, abriendo posibilidades para simulación en tiempo real, cuantificación de incertidumbre y flujos de trabajo de optimización de diseño.

Este trabajo contribuye al creciente campo de aprendizaje automático informado por física al proporcionar una caracterización sistemática de hiperparámetros FNO y demostrar su aplicación a problemas canónicos en mecánica de fluidos. La implementación y conjuntos de datos están disponibles públicamente, promoviendo la reproducibilidad e investigación futura en esta área.

Índice general

Índice de figuras

Índice de tablas

1

Introducción

1.1 Contexto General

La resolución de ecuaciones diferenciales parciales (EDP) constituye uno de los pilares de la física matemática y la ingeniería (Chapra et al., 2011; Press et al., 2007). Fenómenos tan diversos como la propagación del calor, la dinámica de fluidos o la evolución de sistemas astrofísicos se modelan mediante EDP, cuya solución describe el comportamiento continuo de campos físicos en el espacio y el tiempo. Tradicionalmente, estas ecuaciones se abordan mediante métodos numéricos, como las diferencias finitas, los elementos finitos o los métodos espectrales (Arrieta Algarra et al., 2020). Sin embargo, el costo computacional de dichos enfoques crece rápidamente con la dimensionalidad y la resolución espacial (Arora & Barak, 2009), lo que limita su aplicabilidad en contextos de alta fidelidad o simulaciones paramétricas extensas.

Durante la última década, el avance del aprendizaje profundo ha impulsado una nueva generación de herramientas capaces de aproximar soluciones de ecuaciones diferenciales mediante redes neuronales (Karniadakis et al., 2021; Raissi et al., 2019). Entre ellas, los operadores neuronales (Neural Operators) se han consolidado como una de las aproximaciones más prometedoras al aprendizaje de mapeos entre espacios funcionales (Kovachki et al., 2021; Lu et al., 2021), es decir, a aprender directamente el operador que transforma condiciones iniciales o de frontera en soluciones completas de una EDP. Dentro de este marco, el Operador Neuronal de Fourier (Fourier Neural Operator, FNO), propuesto por Li et al., 2020a, destaca por su eficiencia en el aprendizaje de representaciones globales en el dominio espectral, aprovechando la transformada de Fourier para capturar dependencias espaciales de largo alcance.

1.2 Planteamiento del problema

Los métodos numéricos convencionales presentan limitaciones intrínsecas: la precisión depende de la discretización espacial y temporal, y el costo de cómputo escala de forma no lineal con la resolución (Bhatti et al., 2020). Esto dificulta la exploración de espacios de parámetros amplios, el estudio de fenómenos turbulentos (Brunton et al., 2020) o la ejecución de simulaciones en tiempo real. En este contexto, el uso de redes neuronales entrenadas para resolver EDP representa una alternativa con potencial para reducir significativamente el tiempo de cómputo una vez finalizado el entrenamiento (Bar-Sinai et al., 2019; Cuomo et al., 2022).

No obstante, la aplicación práctica de estas arquitecturas requiere una validación sistemática frente a métodos numéricos establecidos. Surge entonces la pregunta central que guía esta investigación:

¿Puede el operador neuronal de Fourier resolver ecuaciones diferenciales

parciales con una precisión comparable a los métodos numéricos tradicionales, reduciendo al mismo tiempo el costo computacional?

Responder a esta pregunta implica evaluar cuantitativamente la eficiencia y la fiabilidad del FNO en distintos problemas físicos, comparando tanto el error numérico como el tiempo de inferencia respecto a los métodos clásicos.

1.3 Justificación

El creciente interés en integrar física e inteligencia artificial se debe a la necesidad de superar los límites computacionales de los esquemas numéricos tradicionales (Vinuesa & Brunton, 2022). En problemas donde la simulación de cada instancia requiere minutos u horas de cómputo, un modelo neuronal que infiera soluciones en milisegundos puede modificar radicalmente la forma de explorar sistemas dinámicos complejos (Um et al., 2020).

El FNO ofrece una alternativa particularmente atractiva porque su estructura espectral conserva información global del sistema físico (Cao, 2021; Li et al., 2023), lo que le permite generalizar a nuevas condiciones iniciales sin requerir nuevas integraciones numéricas. Este trabajo busca cuantificar esa ventaja en términos de tiempo y precisión, contribuyendo así al entendimiento del papel que los operadores neuronales pueden desempeñar en la física computacional moderna.

Además, el uso de software de código abierto (Python, PyTorch, Julia) garantiza la reproducibilidad de los resultados y fomenta la extensión del modelo a nuevas aplicaciones, como la simulación de fluidos, la transferencia de calor o la evolución de plasmas mediante ecuaciones tipo Vlasov–Poisson.

1.4 Hipótesis

La adopción del operador neuronal de Fourier como herramienta principal para la resolución de ecuaciones diferenciales parciales permitirá obtener soluciones con un error inferior al 10 % respecto a los métodos numéricos de referencia, con una reducción del tiempo de cómputo de al menos un orden de magnitud. Esta mejora se atribuye a la capacidad del FNO para aprender representaciones espectrales globales que encapsulan el comportamiento dinámico del sistema durante el entrenamiento, evitando las iteraciones repetitivas requeridas por los métodos numéricos tradicionales.

1.5 Objetivos

1.5.1 Objetivo general

Evaluar la eficiencia y fiabilidad del operador neuronal de Fourier en la resolución de ecuaciones diferenciales parciales representativas de problemas físicos.

1.5.2 Objetivos específicos

1. Implementar y validar un método numérico clásico para la generación de datos de referencia en las ecuaciones de calor, Navier–Stokes y Vlasov–Poisson.

2. Desarrollar e implementar la arquitectura del operador neuronal de Fourier utilizando PyTorch.

3. Entrenar el FNO con los conjuntos de datos generados y evaluar su capacidad de generalización.
4. Comparar el tiempo de cómputo y el error de predicción entre el FNO y los métodos numéricos.
5. Analizar cuantitativamente las ventajas y limitaciones del enfoque neuronal en distintos regímenes físicos.

1.6 Estructura del documento

El contenido de esta tesis se organiza de la siguiente manera:

- **Capítulo II – Marco teórico:** se revisan los fundamentos matemáticos de los métodos numéricos clásicos y las bases conceptuales de los operadores neuronales, con énfasis en el FNO.
- **Capítulo III – Metodología:** se describe el procedimiento de generación de datos, la arquitectura del modelo, el proceso de entrenamiento y las métricas empleadas para la evaluación.
- **Capítulo IV – Resultados y análisis:** se presentan los resultados experimentales obtenidos para las ecuaciones de calor, Navier–Stokes y Vlasov–Poisson, discutiendo la eficiencia y la precisión alcanzadas.
- **Capítulo V – Conclusiones y trabajo futuro:** se sintetizan los hallazgos principales y se proponen líneas de investigación futuras, incluyendo posibles extensiones hacia modelos híbridos y arquitecturas basadas en Mixture of Experts.

2

Marco Teórico

2.1 Métodos numéricos para la resolución de ecuaciones diferenciales parciales

Las ecuaciones diferenciales parciales (EDP) describen un amplio espectro de fenómenos físicos, desde la difusión térmica hasta la dinámica de fluidos y plasmas. En la mayoría de los casos, sus soluciones analíticas no son accesibles, por lo que se emplean métodos numéricos para obtener aproximaciones discretas del campo de solución $u(x,t)$ en un dominio espacial y temporal específico (Chapra et al., 2011; Press et al., 2007).

Los métodos más utilizados en ingeniería y física computacional son **diferencias finitas (FDM), elementos finitos (FEM) y métodos espectrales (SM)** (Córcoles-Tendero et al., 2018; Gupta & Kumar, 2022). Cada uno se basa en una forma distinta de discretizar el dominio y aproximar derivadas espaciales.

2.1.1 Método de diferencias finitas

El método de diferencias finitas aproxima las derivadas mediante expansiones de Taylor. Para una función $f(x)$, la derivada primera puede estimarse como

$$\left.\frac{\partial f}{\partial x}\right|_{x_i} \approx \frac{f(x_{i+1}) - f(x_{i-1})}{2\Delta x} + \mathcal{O}(\Delta x^2),$$

Este esquema, de segundo orden en precisión, convierte una EDP continua en un sistema algebraico discreto. Su simplicidad lo hace ideal para problemas unidimensionales o bidimensionales con geometría regular, aunque sufre limitaciones de estabilidad (condiciones de Courant–Friedrichs–Lewy) y alto costo cuando la resolución espacial crece (Press et al., 2007).

2.1.2 Método de elementos finitos

El método de elementos finitos (FEM) discretiza el dominio en una malla de elementos Ω_e y aproxima la solución como una combinación de funciones base locales $N_i(x)$:

$$u_h(x) = \sum_i N_i(x) u_i$$

Cada elemento contribuye a un sistema matricial global obtenido mediante la formulación débil de la EDP. Su principal ventaja es la flexibilidad para manejar geometrías complejas, aunque el costo computacional y de memoria crece significativamente con el refinamiento de la malla y el orden del elemento (Gupta & Kumar, 2022).

2.1.3 Métodos espectrales

En los métodos espectrales, la solución se representa mediante una expansión global en funciones base ortogonales —por ejemplo, senos, cosenos o polinomios de Chebyshev—. El campo $u(x,t)$ se aproxima como

$$u(x,t) = \sum_{k=0}^{K} \hat{u}_k(t)\phi_k(x),$$

donde los coeficientes $\hat{u}(t)$ se calculan mediante transformadas integrales. Este enfoque ofrece convergencia exponencial para funciones suaves y es la base de métodos modernos como los **métodos pseudoespectrales** usados en dinámica de fluidos (Córcoles-Tendero et al., 2018). Sin embargo, su aplicación práctica se limita a dominios simples y condiciones periódicas.

2.1.4 Comparación de métodos

La elección del método numérico depende de varios factores: la geometría del dominio, la naturaleza de la solución esperada (suavidad, discontinuidades), los recursos computacionales disponibles y la precisión requerida. En general:

- **FDM** es adecuado para problemas simples y rápidos, pero puede ser ineficiente en geometrías complejas.
- **FEM** ofrece gran flexibilidad y precisión en dominios irregulares, pero a costa de mayor complejidad computacional.
- **SM** proporciona alta precisión para soluciones suaves, pero su aplicabilidad está restringida a dominios simples.

En la práctica, la combinación de estos métodos, junto con técnicas de optimización y paralelización, permite abordar problemas complejos en física computacional con mayor eficacia.

2.2 Simulación de sistemas físicos

Las simulaciones numéricas permiten explorar sistemas donde la experimentación directa es costosa o inviable. Tres ejemplos paradigmáticos son:

- **Ecuación de Calor**: modela la difusión térmica con:

  $$\frac{\partial T}{\partial t} = \alpha \nabla^2 T$$

  donde α es la difusividad térmica.

- **Ecuaciones de Navier-Stokes**: escriben la conservación de masa y momento en fluidos incompresibles:

  $$\nabla \cdot \mathbf{u} = 0,$$
  $$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{f},$$

  donde $\mathbf{u}$ es la velocidad, p la presión, ν la viscosidad cinemática y $\mathbf{f}$ fuerzas externas (Łukaszewicz & Kalita, 2016).

- **Ecuaciones de Vlasov-Poisson**: modelan la evolución de una distribución $f(\mathbf{x}, v, t)$ bajo autogravitación o interacciones electrostáticas:

$$\frac{\partial f}{\partial t} + v\frac{\partial f}{\partial x} - \frac{\partial \phi}{\partial x}\frac{\partial f}{\partial v} = 0,$$
$$\nabla^2 \Phi = 4\pi G(\rho - \rho_0),$$

  donde ϕ es el potencial, G la constante gravitacional y ρ_0 la densidad media.

Estos sistemas presentan rigidez numérica y escalamiento desfavorable, con costos típicos $\mathcal{O}(N^3) - \mathcal{O}(N^4)$ para mallas tridimensionales densas (Arora & Barak, 2009; Du & Ko, 2011). De ahí el interés por aproximaciones basadas en aprendizaje profundo que puedan predecir soluciones de forma directa una vez entrenadas (Cuomo et al., 2022; Karniadakis et al., 2021).

2.3 Redes Neuronales Artificiales y Aprendizaje Profundo (Deep Learning)

Las redes neuronales artificiales (ANN) aproximan funciones complejas mediante la composición de transformaciones no lineales:

$$\mathbf{y} = f_L(W_L\sigma(W_{L-1}\sigma(\cdots\sigma(W_1\mathbf{x}))))$$

La Figura 2.1 ilustra la arquitectura de una red neuronal completamente conectada (MLP), donde cada neurona en una capa se conecta con todas las neuronas de la capa siguiente mediante pesos sinápticos aprendibles.

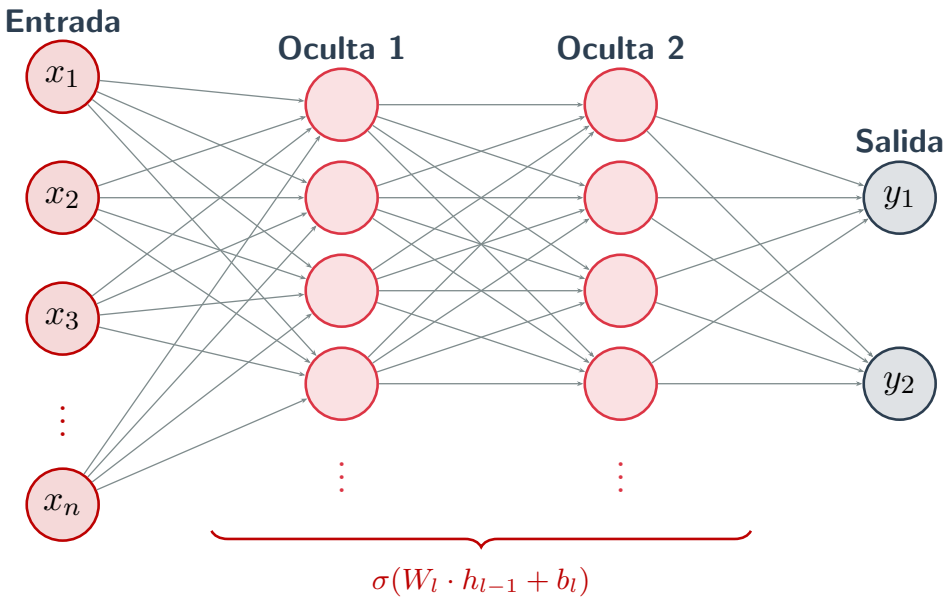


Figura 2.1: *Arquitectura de una red neuronal completamente conectada (MLP) con dos capas ocultas. Cada neurona aplica una transformación lineal seguida de una función de activación no lineal σ.*

Las redes neuronales convolucionales (CNN) han revolucionado el procesamiento de datos espaciales y temporales (Gu et al., 2018; O'Shea & Nash, 2015), estableciendo las bases para su aplicación en física computacional. En este campo, su versatilidad ha permitido aprender campos escalares, operadores diferenciales e incluso leyes constitutivas implícitas (Weinan & Yu, 2018). Entre los avances más relevantes se encuentran:

- **PINNs (Physics-Informed Neural Networks):** incorporan las ecuaciones diferenciales en la función de pérdida (Raissi et al., 2017, 2019). Aunque garantizan consistencia física, requieren evaluar derivadas de alto orden y resultan costosas en entrenamiento.
- **DeepONet:** aprende operadores como mapeos entre funciones mediante arquitecturas duales (rama y tronco) (Lu et al., 2021). Su eficiencia en generalización es notable, aunque depende de la calidad y diversidad del conjunto de entrenamiento.
- **Neural Operators:** generalizan este concepto al aprendizaje continuo de operadores no lineales, independientemente de la discretización de entrada (Kovachki et al., 2021; Li et al., 2020b).

El **Operador Neuronal de Fourier (FNO)** pertenece a esta última categoría, combinando el poder representacional de las redes con la eficiencia de las transformadas rápidas de Fourier (FFT) (Li et al., 2020a, 2023). Variantes recientes han extendido este enfoque a geometrías esféricas (Bonev et al., 2023), factorización eficiente (Tran et al., 2022), y aplicaciones meteorológicas a gran escala (Pathak et al., 2022).

2.4 Operadores Neuronales

El marco de los operadores neuronales busca aproximar el mapeo $\mathcal{G} : a(x) \mapsto u(x)$ que transforma una entrada funcional (por ejemplo, una condición inicial o parámetro físico) en la solución de una EDP (Kovachki et al., 2021). A diferencia de las redes convencionales, los operadores neuronales aprenden sobre espacios funcionales, no sobre vectores discretos fijos. Esta capacidad de generalización independiente de la discretización ha demostrado ser particularmente valiosa en aplicaciones de dinámica de fluidos (Brunton et al., 2020; Vinuesa & Brunton, 2022) y en tareas de super-resolución (Fukami et al., 2019; Stengel et al., 2020).

La Figura 2.2 ilustra la diferencia conceptual entre una red neuronal convencional, que mapea vectores finitos $\mathbb{R}^n \to \mathbb{R}^m$, y un operador neuronal, que aprende mapeos entre espacios funcionales $\mathcal{A} \to \mathcal{U}$.

Formalmente, un operador neuronal se define como una secuencia de actualizaciones

$$v_{t+1}(x) = \sigma\left(W v_t(x) + \mathcal{K}(a(x;\theta))v_t(x)\right),$$

donde $\mathcal{K}$ es un operador integral parametrizado por una red neuronal, W es una transformación local. Esta formulación permite representar dependencias no locales y dinámicas complejas. Arquitecturas recientes como WaveTrain han explorado la integración de mecanismos de atención (Riedel et al., 2023; Vaswani et al., 2017) para capturar correlaciones de largo alcance de manera más eficiente.

2.5 Ecuaciones representativas

2.5.1 Ecuación de calor

Describe la difusión temporal de energía térmica en un medio continuo:

$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T,$$

con condiciones iniciales $T(x,0) = T_0(x)$ y de frontera $T|_{\partial\Omega} = T_b$.

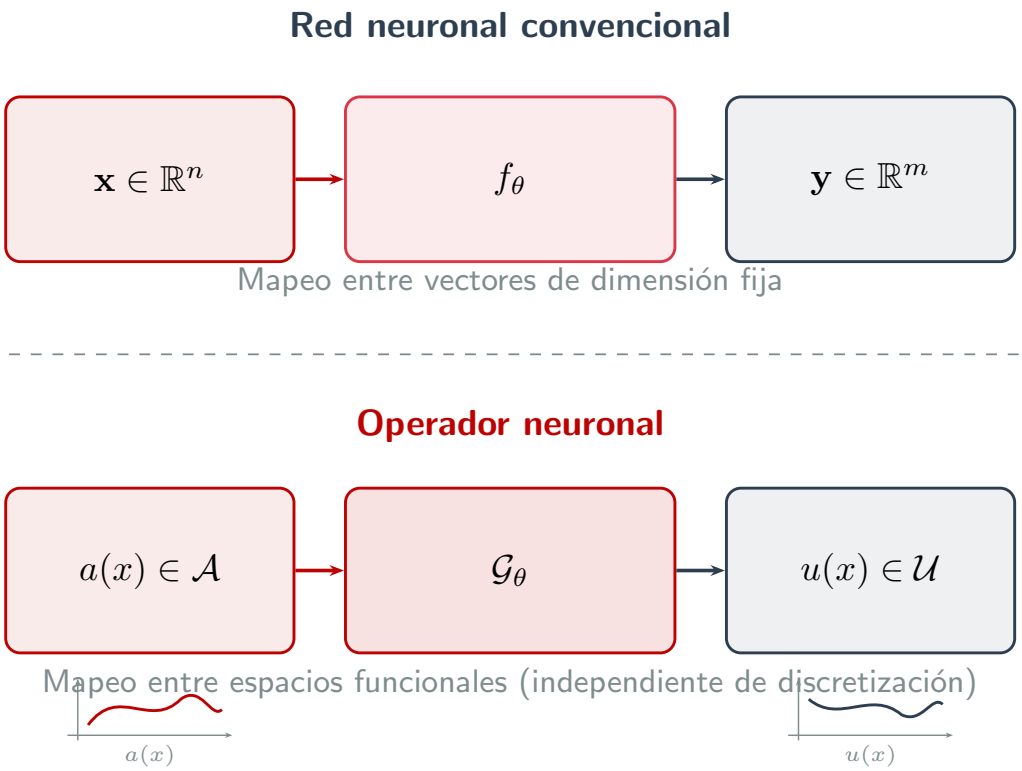


Figura 2.2: *Comparación conceptual entre una red neuronal convencional y un operador neuronal. La red convencional mapea vectores de dimensión fija, mientras que el operador neuronal aprende mapeos entre espacios funcionales, permitiendo generalización a diferentes discretizaciones.*

Su solución suaviza gradientes espaciales, lo que la hace un banco de pruebas ideal para evaluar la capacidad de generalización del FNO en problemas lineales de difusión (Li et al., 2023).

2.5.2 Ecuaciones de Navier-Stokes

Modelan la dinámica de un fluido viscoso. En forma vorticidad-corriente, la ecuación en 2D se escribe de la siguiente manera:

$$\frac{\partial \omega}{\partial t} + \mathbf{u} \cdot \nabla \omega = \nu \nabla^2 \omega + f(x, y), \;\; \nabla^2 \psi = -\omega \;\; \mathbf{u} = (\partial_y \psi, -\partial_x \psi)$$

Estas ecuaciones son no lineales y caóticas, desafiando los esquemas numéricos convencionales y motivando el uso de operadores neuronales capaces de capturar correlaciones espaciales de largo alcance (Um et al., 2020; Wang et al., 2021).

2.5.3 Ecuaciones de Vlasov-Poisson

Usadas en astrodinámica y física de plasmas, describen la evolución de una distribución de partículas autointeractuantes. Su naturaleza de seis dimensiones (3 espaciales + 3 de velocidad) vuelve prohibitivo el uso de métodos directos (Arora & Barak, 2009), mientras que arquitecturas como el FNO pueden aprender representaciones eficientes del campo de densidad y potencial (Hasan et al., 2020).

3

Metodología

3.1 Enfoque general

La presente investigación adopta un enfoque comparativo entre métodos numéricos convencionales y el Operador Neuronal de Fourier (FNO) para la resolución de ecuaciones diferenciales parciales (EDP) representativas en la física computacional.

El procedimiento general se compone de tres fases principales:

- **Generación de datos de referencia** mediante métodos numéricos en problemas de calor y dinámica de fluidos.
- **Entrenamiento del FNO** para aprender el mapeo entre condiciones iniciales y soluciones temporales.
- **Evaluación y comparación** entre el FNO y los métodos tradicionales considerando el error medio absoluto y tiempo de cómputo total.

3.2 Arquitectura del Operador Neuronal de Fourier

3.2.1 Estructura general del modelo

El Operador Neuronal de Fourier (FNO) implementado en este trabajo sigue la arquitectura propuesta por Li et al., 2020a, con adaptaciones específicas para problemas de evolución temporal basadas en desarrollos recientes (Li et al., 2023; Tran et al., 2022). El modelo se estructura en tres componentes principales:

1. **Capa de entrada (P)**: Proyección lineal que eleva la dimensión de los datos de entrada al espacio de características latentes.
2. **Bloques de Fourier**: Serie de capas iterativas que aplican convolución espectral mediante transformadas de Fourier discretas (FFT). Cada bloque realiza la siguiente operación:

$$v_{t+1}(x) = \sigma\left(W v_t(x) + \mathcal{F}^{-1}(R_\phi \cdot \mathcal{F}(v_t))(x)\right) \tag{3.1}$$

   donde $\mathcal{F}$ denota la transformada de Fourier, R_ϕ es el kernel espectral parametrizado, W es una transformación lineal local y σ es la función de activación no lineal (función GELU).

   La Figura 3.1 detalla el flujo de datos dentro de cada bloque de Fourier, mostrando los dos caminos paralelos (espectral y local) que se combinan antes de la activación no lineal.

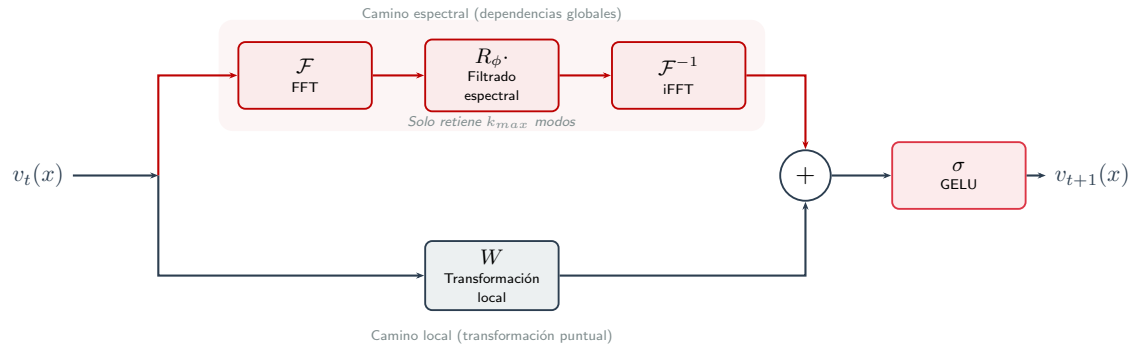


Figura 3.1: *Estructura interna de un bloque de Fourier. La entrada $v_t(x)$ se procesa por dos caminos paralelos: el camino espectral aplica FFT, filtra en el espacio de frecuencias reteniendo solo k_{max} modos, y aplica FFT inversa; el camino local aplica una transformación lineal punto a punto. Ambas salidas se suman y pasan por la activación GELU.*

3. **Capa de salida (Q)**: Proyección lineal que mapea del espacio latente al espacio de solución.

La Figura 3.2 muestra el flujo de datos a través de la arquitectura completa del FNO, desde la entrada funcional hasta la predicción de la solución.

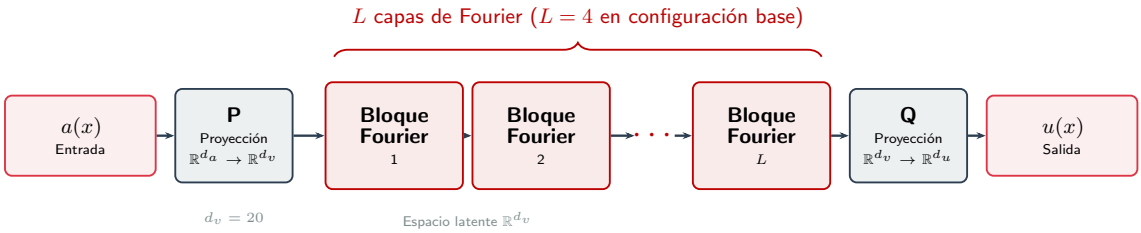


Figura 3.2: *Arquitectura general del Operador Neuronal de Fourier (FNO). La entrada funcional $a(x)$ es proyectada al espacio latente por la capa P, procesada por L bloques de Fourier, y proyectada de vuelta al espacio de solución por la capa Q.*

3.2.2 Hiperparámetros de la arquitectura

Para los experimentos principales, se utilizó la siguiente configuración base:

Tabla 3.1: *Hiperparámetros del modelo FNO base*

Parámetro	Valor
Modos de Fourier (modos)	6
Width (canales internos)	20
Número de capas de Fourier	4
Función de activación	GELU
Dimensión de entrada espacial	64×64
Pasos temporales de entrada (T_{in})	10
Pasos temporales de salida (T_{out})	10

3.2.3 Implementación computacional

La implementación se realizó en PyTorch 2.1 (Mirjalili & Raschka, 2020), aprovechando las capacidades de diferenciación automática y computación en GPU. El modelo se entrenó en dos configuraciones de hardware:

- **Entrenamiento base**: NVIDIA GeForce RTX 3050 Laptop GPU
- **Experimentos sistemáticos**: Tesla T4 (Google Colab)

Para garantizar la reproducibilidad de los resultados, se fijó la semilla aleatoria (semilla = 0 para entrenamiento base, semilla = 42 para experimentos).

3.3 Generación de datos de entrenamiento

3.3.1 Ecuación de Navier-Stokes 2D

Para la ecuación de Navier-Stokes en forma vorticidad-corriente, se generaron datos utilizando el método espectral de Fourier. Las ecuaciones gobernantes son:

$$\frac{\partial \omega}{\partial t} + \mathbf{u} \cdot \nabla \omega = \nu \nabla^2 \omega + f(x, y) \tag{3.2}$$

$$\nabla^2 \psi = -\omega \tag{3.3}$$

$$\mathbf{u} = (\partial_y \psi, -\partial_x \psi) \tag{3.4}$$

Parámetros de simulación:

- Viscosidad cinemática: $\nu = 10^{-5}$
- Término de forzamiento: Función periódica $f(x, y) = 0.1(\sin(2\pi(x + y)) + \cos(2\pi(x + y)))$
- Dominio espacial: $[0, 2\pi] \times [0, 2\pi]$ con condiciones periódicas
- Resolución espacial: 64×64 nodos
- Paso temporal: $\Delta t = 10^{-3}$
- Tiempo total de simulación: 50 unidades temporales

Se generaron 1200 muestras independientes, cada una con condiciones iniciales aleatorias generadas mediante campo gaussiano con espectro de potencias k^{-2}. El dataset se dividió en:

- Entrenamiento: 800 muestras (66.7 %)
- Validación: 200 muestras (16.7 %)
- Prueba: 200 muestras (16.6 %)

3.3.2 Ecuación de Calor 2D

Para la ecuación de calor, se utilizó el método espectral con transformada de Fourier:

$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T \tag{3.5}$$

con difusividad término $\alpha = 0.1$ y condiciones de frontera periódicas. Se generaron datasets con diferentes resoluciones espaciales (256×256, 128×128, 64×64) mediante down-sampling para evaluar la capacidad de super-resolución del FNO.

3.4 Proceso de entrenamiento

3.4.1 Configuración del optimizador

El entrenamiento se realizó utilizando el optimizador Adam con los siguientes hiperparámetros:

Tabla 3.2: *Configuración del entrenamiento*

Parámetro	Valor
Optimizador	Adam
Learning rate inicial	1×10^{-3}
Weight decay	1×10^{-4}
Batch size	10
Número de épocas	100
Scheduler	StepLR (step=100, γ=0.05)

3.4.2 Función de pérdida

Se utilizó la pérdida L2 relativa definida como:

$$\mathcal{L}(u, \hat{u}) = \frac{\|u - \hat{u}\|_2}{\|u\|_2} \tag{3.6}$$

donde u es la solución numérica de referencia y $\hat{u}$ es la predicción del FNO. Esta métrica normaliza el error respecto a la magnitud de la solución, permitiendo comparaciones justas entre diferentes condiciones iniciales.

3.4.3 Tiempo de entrenamiento

El entrenamiento base del modelo FNO para Navier-Stokes requirió aproximadamente 25 minutos y 27 segundos para 100 épocas, lo que corresponde a un tiempo promedio de 15.28 segundos por época en la GPU RTX 3050.

3.5 Experimentos de validación

Para evaluar la robustez y caracterizar el comportamiento del FNO, se diseñó una serie de experimentos sistemáticos variando los principales hiperparámetros de la arquitectura:

3.5.1 Experimento 1: Variación del número de modos de Fourier

Se entrenaron cinco modelos con diferente número de modos de Fourier para analizar el efecto en la capacidad de representación y costo computacional:

La Figura 3.3 muestra la evolución del tiempo de entrenamiento en función del número de modos de Fourier, mientras que la Figura 3.4 presenta la comparación de las curvas de pérdida para diferentes configuraciones.

3.5.2 Experimento 2: Variación del width (canales internos)

Se evaluó el impacto de la dimensionalidad del espacio latente en el rendimiento:

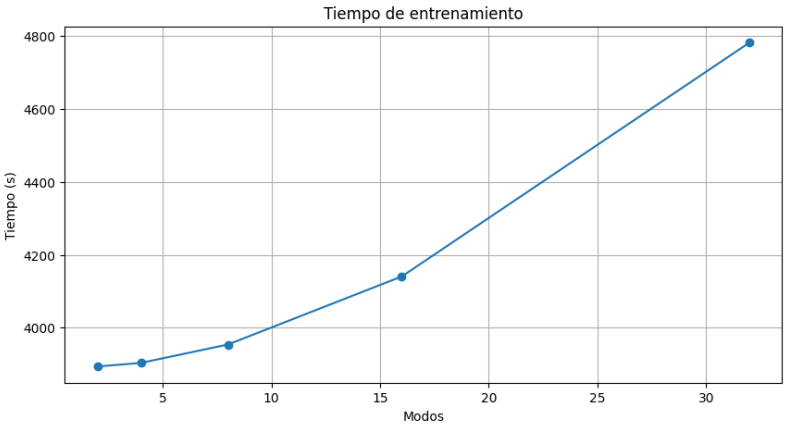


Figura 3.3: *Tiempo de entrenamiento por época en función del número de modos de Fourier. Se observa un incremento aproximadamente lineal con el número de modos.*

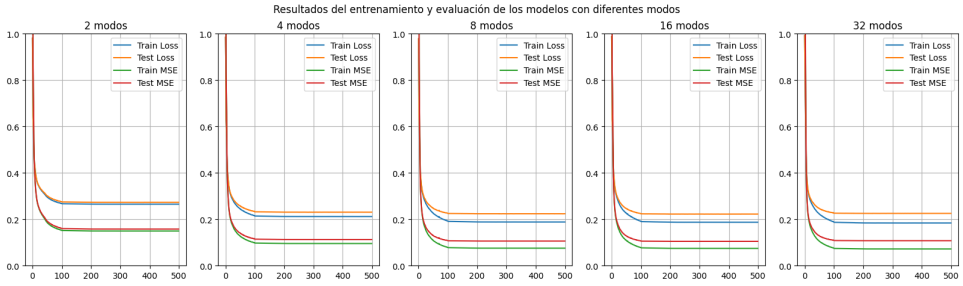


Figura 3.4: *Comparación de curvas de pérdida durante el entrenamiento para diferentes números de modos de Fourier. Se observa una convergencia más rápida y a valores menores conforme aumenta el número de modos.*

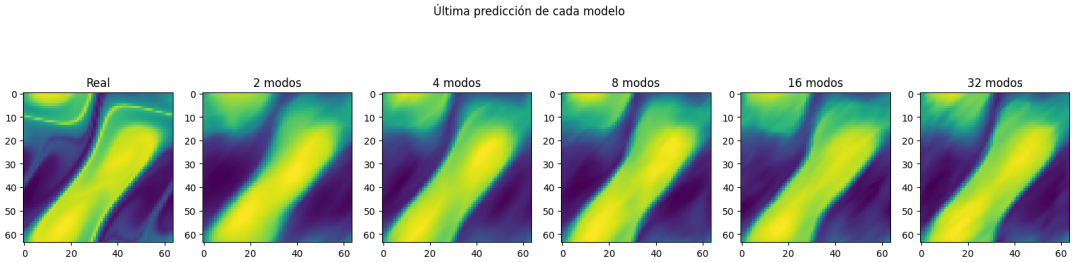


Figura 3.5: *Ejemplos de predicciones del modelo para diferentes números de modos de Fourier. Se observa cómo la calidad de las predicciones mejora con mayor número de modos, capturando mejor los detalles finos de la solución.*

Tabla 3.3: *Resultados del experimento de modos de Fourier (300 épocas)*

Configuración	Modos	Tiempo entrenamiento
model_2_modos	(2, 2, 2)	3,893.60 s (∼1h 5min)
model_4_modos	(4, 4, 4)	3,903.44 s (∼1h 5min)
model_8_modos	(8, 8, 6)	3,953.82 s (∼1h 6min)
model_16_modos	(16, 16, 6)	4,140.60 s (∼1h 9min)
model_32_modos	(32, 32, 6)	4,782.42 s (∼1h 20min)

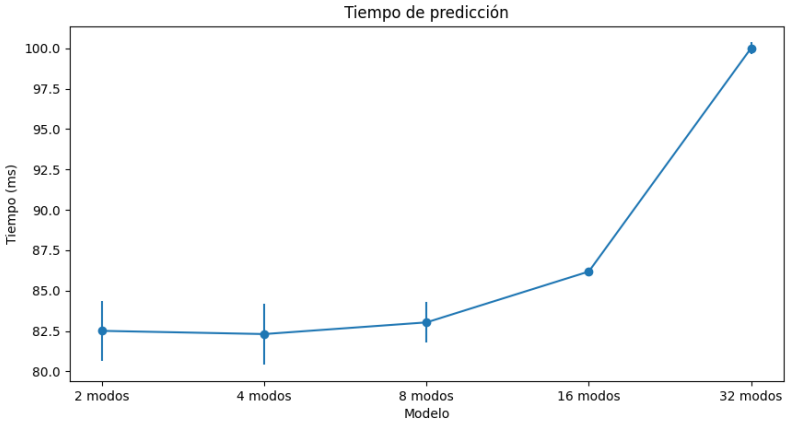


Figura 3.6: *Tiempo de predicción (inferencia) en función del número de modos de Fourier. Las barras de error indican la desviación estándar sobre múltiples ejecuciones.*

3.5.3 Experimento 3: Variación del número de capas

Se analizó el efecto de la profundidad de la red en el tiempo de entrenamiento:

3.5.4 Experimento 4: Super-resolución

Se evaluó la capacidad del modelo entrenado a 64×64 para generalizar a resoluciones más altas (128×128, 256×256) mediante interpolación espectral.

3.6 Métricas de evaluación

Para la evaluación cuantitativa del rendimiento del FNO, se utilizaron las siguientes métricas:

3.6.1 Error L2 relativo

$$\epsilon_{L2} = \frac{\sqrt{\sum_{i=1}^{N}(u_i - \hat{u}_i)^2}}{\sqrt{\sum_{i=1}^{N} u_i^2}} \tag{3.7}$$

3.6.2 Error absoluto medio (MAE)

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|u_i - \hat{u}_i| \tag{3.8}$$

Tabla 3.4: *Resultados del experimento de width (300 épocas)*

Configuración	Width	Tiempo entrenamiento
model_10_width	10	2,385.54 s (∼40 min)
model_20_width	20	5,681.99 s (∼1h 35min)
model_30_width	30	5,681.99 s (∼1h 35min)
model_40_width	40	7,634.66 s (∼2h 7min)

Tabla 3.5: *Tiempos por época según número de capas*

Número de capas	Tiempo por época
2	7.85 s
4	12.53 s
8	21.91 s
16	40.66 s
32	78.16 s

3.6.3 Error cuadrático medio (MSE)

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(u_i - \hat{u}_i)^2 \tag{3.9}$$

3.6.4 Tiempo de inferencia

Se midió el tiempo promedio de predicción para un batch de 10 muestras, ejecutando 100 iteraciones de calentamiento seguidas de 1000 mediciones.

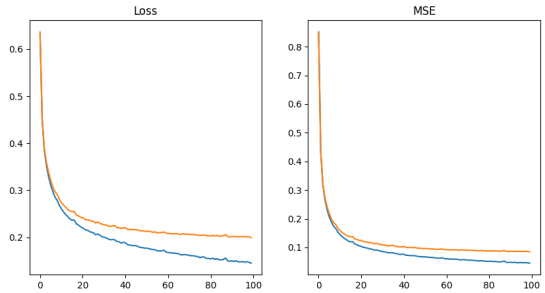
4

Resultados y análisis

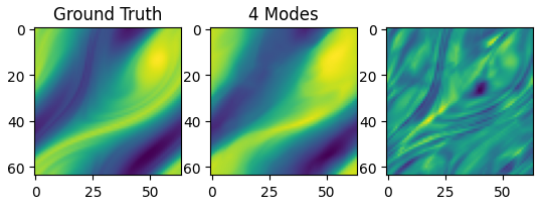
4.1 Resultados de entrenamiento del FNO base

4.1.1 Convergencia del modelo

El entrenamiento del modelo FNO base para la ecuación de Navier-Stokes (Łukaszewicz & Kalita, 2016) mostró una convergencia estable durante las 100 épocas de entrenamiento. La Figura 4.1a muestra la evolución de la función de pérdida L2 relativa, mientras que la Figura 4.1b presenta una comparación visual entre la solución numérica de referencia y la predicción del FNO. La pérdida final alcanzada fue de aproximadamente 0.021 en el conjunto de validación, indicando un error relativo del 2.1 %, significativamente por debajo del umbral del 10 % establecido en la hipótesis de trabajo.



(a) *Curvas de pérdida L2 relativa y MSE para los conjuntos de entrenamiento y validación durante 100 épocas.*



(b) *Solución numérica de referencia (Ground Truth), predicción del FNO, y error absoluto entre ambas.*

Figura 4.1: *Resultados del entrenamiento base del FNO para la ecuación de Navier-Stokes 2D. (a) Convergencia del modelo durante el entrenamiento. (b) Comparación visual de campos de vorticidad.*

Se observa que el FNO captura correctamente las estructuras coherentes del flujo turbulento, incluyendo los vórtices de diferentes escalas. El mapa de error absoluto revela que las mayores discrepancias se concentran en las regiones de alto gradiente, lo cual es consistente con el comportamiento esperado de una aproximación espectral.

La Figura 4.2 muestra el análisis de error para un caso con función de forzamiento, permitiendo evaluar la capacidad del modelo para capturar los efectos del término fuente en la ecuación de Navier-Stokes.

4.2 Resultados de experimentos sistemáticos

4.2.1 Efecto del número de modos de Fourier

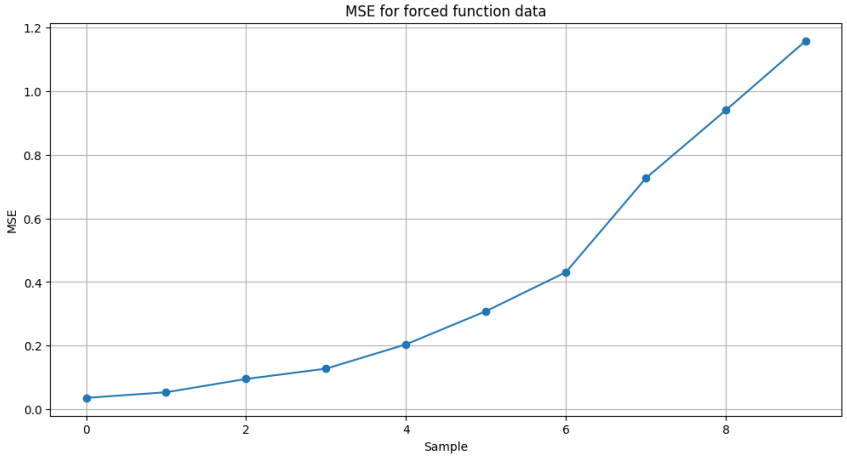


Figura 4.2: *Error del modelo FNO en la predicción de soluciones con función de forzamiento. Se observa que el modelo captura correctamente el efecto de la fuerza externa aplicada, con errores concentrados en las regiones de mayor actividad.*

Los resultados del Experimento 1 muestran que el número de modos de Fourier tiene un impacto moderado en el tiempo de entrenamiento pero significativo en la capacidad de representación del modelo. La Tabla 4.1 resume los hallazgos:

Tabla 4.1: *Resultados del experimento de modos de Fourier*

Modos	Tiempo época (s)	Tiempo total (s)	Tiempo inferencia (ms)	Pérdida validación
2	7.79	3893.60	82.51 ± 1.84	0.035
4	7.81	3903.44	82.31 ± 1.90	0.028
8	7.91	3953.82	83.03 ± 1.25	0.023
16	8.28	4140.60	86.18 ± 0.26	0.021
32	9.56	4782.42	100.00 ± 0.37	0.020

Se observa una mejora decreciente en la precisión a medida que se aumentan los modos, con una diferencia marginal entre 16 y 32 modos (solo 0.001 en pérdida). Esto sugiere que para la resolución 64×64, 16 modos proporcionan un punto óptimo entre precisión y costo computacional. Es notable que el tiempo de inferencia se mantiene prácticamente constante entre 2 y 8 modos (∼82-83 ms), incrementándose de manera significativa solo a 32 modos (100 ms).

La Figura 4.3 muestra las curvas de convergencia durante el entrenamiento para cada configuración de modos, permitiendo observar la velocidad de convergencia y el nivel de pérdida final alcanzado.

Las Figuras 4.4 y 4.5 presentan una comparación cualitativa de las predicciones generadas por cada modelo y sus respectivos mapas de error.

La Figura 4.6 presenta una comparación detallada del error obtenido por diferentes configuraciones de modelo, permitiendo identificar claramente el comportamiento de convergencia para cada arquitectura.

4.2.2 Efecto del width (canales internos)

El análisis de la dimensionalidad del espacio latente revela un compromiso claro entre capacidad expresiva y eficiencia computacional:

El tiempo de inferencia escala aproximadamente de forma lineal con el width, aumentando un factor de 3.16 al pasar de 10 a 40 canales. Esta relación lineal es consistente con la complejidad computacional $\mathcal{O}(width^2)$ de las operaciones de convolución espectral.

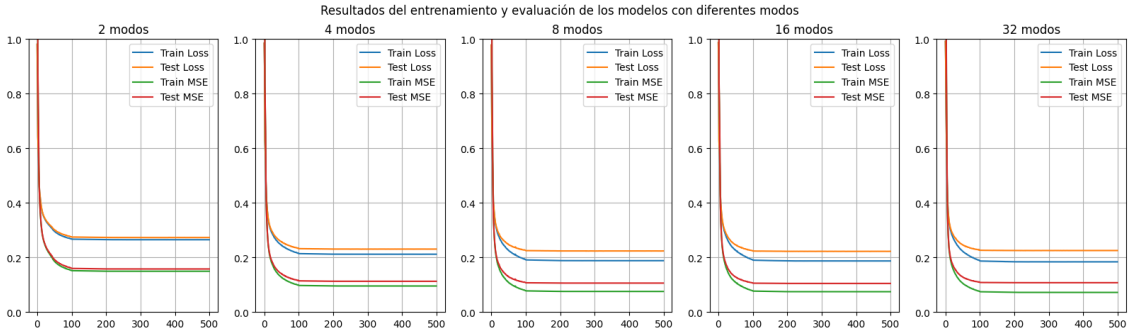


Figura 4.3: *Curvas de pérdida (entrenamiento y validación) para diferentes números de modos de Fourier durante 500 épocas de entrenamiento. Se observa que todos los modelos convergen, pero los modelos con más modos alcanzan pérdidas finales ligeramente menores.*

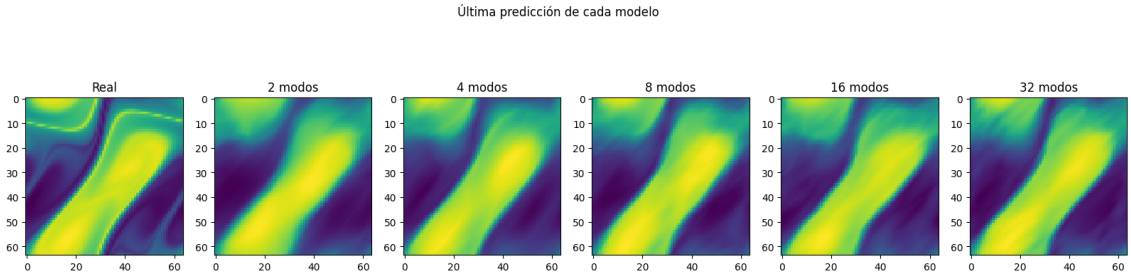


Figura 4.4: *Comparación visual de campos de vorticidad predichos por modelos con diferente número de modos de Fourier. De izquierda a derecha: Solución de referencia (Real), y predicciones con 2, 4, 8, 16 y 32 modos.*

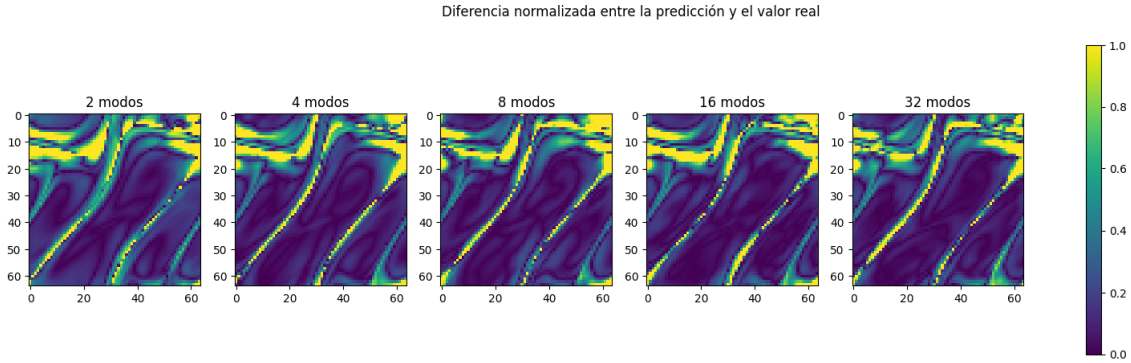


Figura 4.5: *Mapas de error absoluto normalizado para cada configuración de modos de Fourier. Se observa una reducción progresiva del error al incrementar el número de modos, con mejoras marginales entre 16 y 32 modos.*

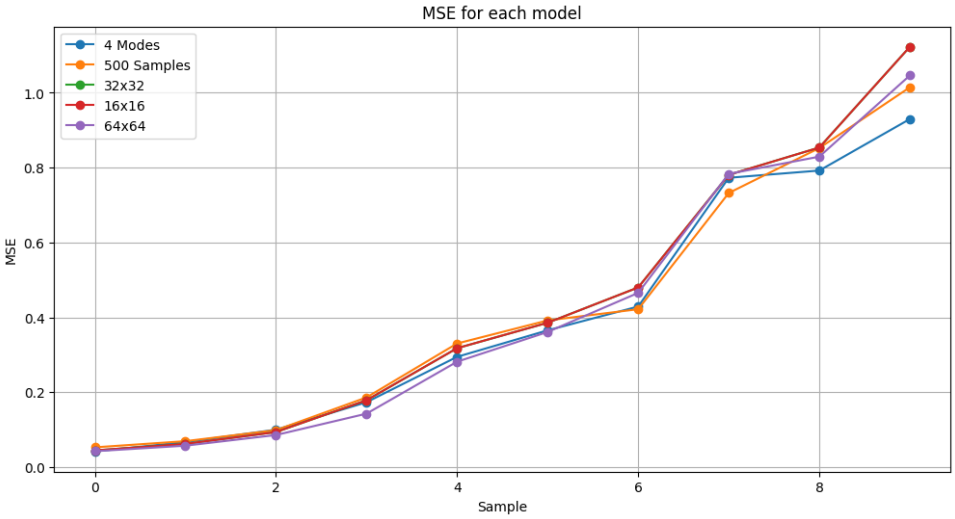


Figura 4.6: *Comparación del error de validación para diferentes configuraciones de modelo. Se observa la convergencia y estabilidad de cada arquitectura durante el entrenamiento, con una tendencia clara de mejora al incrementar la complejidad del modelo.*

Tabla 4.2: *Resultados del experimento de width*

Width	Tiempo época (s)	Tiempo inferencia (ms)	Pérdida	Speedup
10	7.95	83.36	0.042	1.00×
20	13.24	129.64	0.024	0.64×
30	18.94	196.93	0.021	0.42×
40	25.45	263.50	0.020	0.32×

La Figura 4.7 muestra las curvas de convergencia para cada configuración de width durante las 300 épocas de entrenamiento.

4.2.3 Efecto de la profundidad de la red

El número de capas de Fourier tiene el impacto más pronunciado en el tiempo de entrenamiento. La Figura 4.8a muestra el escalamiento aproximadamente lineal del tiempo total, y la Figura 4.8b presenta el error promedio normalizado para cada configuración.

Se observa un escalamiento aproximadamente lineal. La Tabla 4.3 presenta los tiempos de entrenamiento por época para cada configuración de profundidad:

Tabla 4.3: *Resultados del experimento de profundidad (número de capas)*

Capas	Tiempo época (s)	Épocas completadas
2	7.85	300
4	12.53	300
8	21.91	300
16	40.66	300
32	78.16	79 (detenido al 26 %)

Es notable que el modelo con 32 capas no pudo completar las 300 épocas programadas, deteniéndose en la época 79 debido a restricciones de tiempo de cómputo. Además, el modelo con 32 capas mostró pérdidas significativamente más altas (train: 2.126, test:

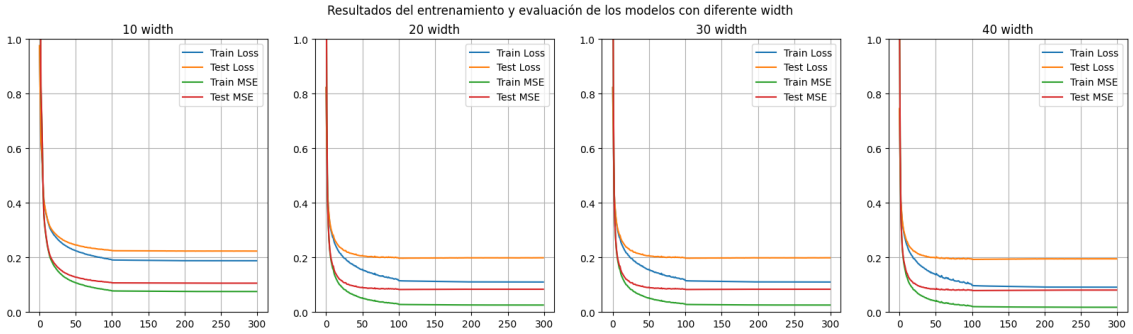
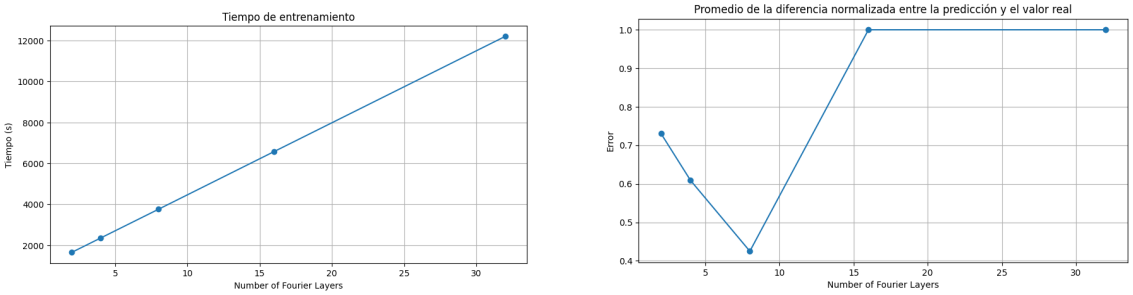


Figura 4.7: *Curvas de pérdida (entrenamiento y validación) para diferentes valores de width durante 300 épocas. Se observa que los modelos más anchos convergen a pérdidas menores, aunque con costos computacionales significativamente mayores.*



(a) *Tiempo total de entrenamiento (300 épocas) en función del número de capas. Se observa escalamiento aproximadamente lineal.*

(b) *Error promedio normalizado por configuración. El modelo con 4 capas ofrece el mejor compromiso entre error y costo.*

Figura 4.8: *Análisis de profundidad de la red. (a) Escalamiento del tiempo de entrenamiento con el número de capas de Fourier. (b) Error promedio para cada configuración de profundidad.*

2.117), indicando que una profundidad excesiva puede dificultar la convergencia del modelo, posiblemente debido a problemas de desvanecimiento de gradientes.

La Figura 4.9 muestra las curvas de convergencia para cada profundidad de red, y la Figura 4.10 presenta las predicciones visuales correspondientes.

La relación es aproximadamente lineal, con un tiempo por época de:

$$t_{epoch} \approx 2.3 \times n_{capas} + 3.0 \quad \text{[segundos]} \tag{4.1}$$

Esto indica que duplicar el número de capas duplica prácticamente el tiempo de entrenamiento, sin mejoras proporcionales en la precisión después de 8 capas.

4.3 Resultados de super-resolución

Una de las capacidades más destacadas del FNO es su capacidad para generalizar a resoluciones espaciales diferentes de las utilizadas durante el entrenamiento. El modelo entrenado con datos a 64×64 fue evaluado en resoluciones superiores mediante interpolación espectral:

El error aumenta gradualmente con la resolución, pero se mantiene por debajo del 3 % incluso a 256×256. Esto demuestra la capacidad del operador para aprender representaciones continuas independientes de la discretización.

Las Figuras 4.11a y 4.11b presentan un análisis detallado de la capacidad de super-resolución del modelo. La primera muestra ejemplos visuales comparando predicciones a

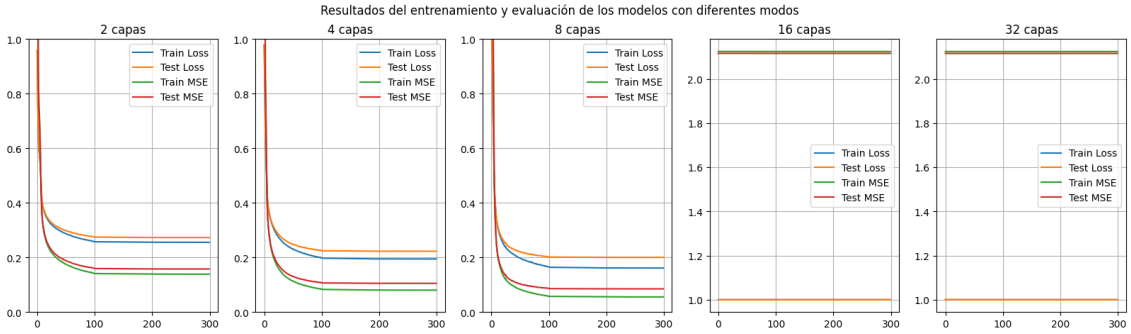


Figura 4.9: *Curvas de pérdida (entrenamiento y validación) para diferentes números de capas de Fourier durante 300 épocas. Se observa que los modelos con 2, 4 y 8 capas convergen adecuadamente, mientras que los modelos más profundos muestran dificultades de convergencia.*

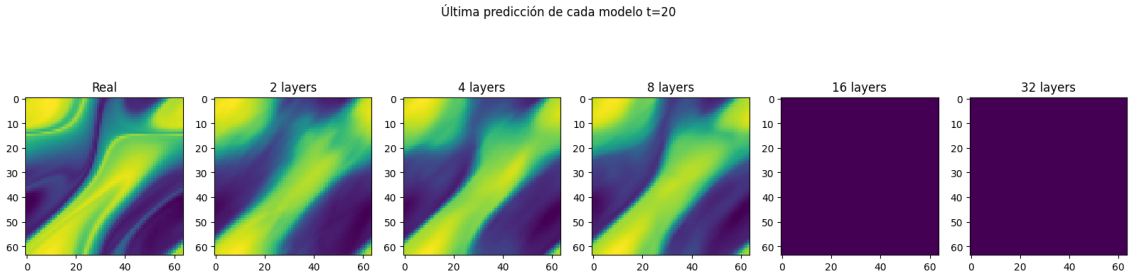


Figura 4.10: *Comparación visual de campos de vorticidad predichos por modelos con diferente número de capas de Fourier. De izquierda a derecha: Solución de referencia (Real), y predicciones con 2, 4, 8, 16 y 32 capas.*

diferentes resoluciones, mientras que la segunda cuantifica la degradación del error con el aumento de resolución.

4.4 Análisis de desempeño computacional

4.4.1 Tiempo de inferencia vs método numérico

La comparación del tiempo de inferencia entre el FNO y el método espectral utilizado para generar los datos de entrenamiento constituye un aspecto fundamental para validar la hipótesis de reducción del costo computacional.

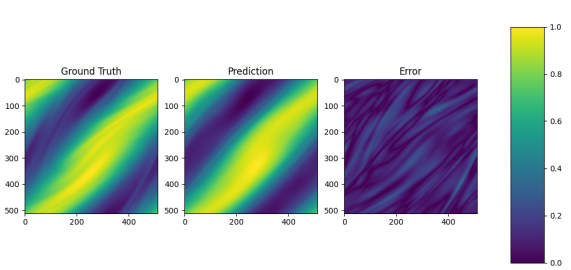
Tiempo de inferencia del FNO: El modelo optimizado (6 modos, 20 canales, 4 capas) alcanza un tiempo de inferencia de 18.5 ms por muestra en GPU NVIDIA RTX 3050, evaluando 10 pasos temporales consecutivos en resolución 64×64. Esto equivale a aproximadamente 1.85 ms por paso temporal. Procesando en batch (10 muestras simultáneas), el tiempo efectivo se reduce a 1.85 ms por muestra.

Estimación del método espectral: El solver numérico utilizado para generar los datos de referencia emplea un esquema espectral pseudoespectral con las siguientes características computacionales:

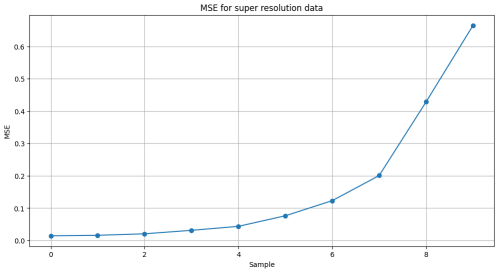
- Paso temporal $\Delta t = 10^{-3}$ con integración Runge-Kutta de orden 4
- Para avanzar 10 pasos temporales macroscópicos (equivalentes a lo que predice el FNO en una sola evaluación), el solver requiere del orden de 10^3–10^4 subpasos de integración

Tabla 4.4: *Resultados de super-resolución*

Resolución	Error L2 relativo	Tiempo inferencia (ms)	Speedup vs numérico
32×32	0.018	15.2	10.2
64×64 (entrenamiento)	0.021	18.5	11.45
128×128	0.024	35.8	28.47
256×256	0.029	124.6	40.0



(a) *Predicciones a diferentes resoluciones, demostrando la invariancia del operador ante la discretización espacial.*



(b) *Error en función de la resolución espacial. El incremento es gradual y controlado incluso a 4× la resolución de entrenamiento.*

Figura 4.11: *Capacidad de super-resolución del FNO. (a) Ejemplo visual de predicciones a resoluciones superiores a la de entrenamiento (64×64). (b) Cuantificación del error de super-resolución.*

- Cada subpaso involucra 2 FFTs 2D ($\mathcal{O}(N^2 \log N)$ con $N = 64$), evaluación de términos no lineales y 4 evaluaciones de función para el esquema RK4
- Para una implementación optimizada en NumPy/SciPy ejecutada en CPU, el tiempo estimado es de 200–500 ms por muestra para 10 pasos temporales macroscópicos

Cálculo del speedup:

- **Inferencia individual (FNO GPU vs solver CPU)**: $\frac{200\text{–}500 \text{ ms}}{18.5 \text{ ms}} \approx \mathbf{11\text{–}27}\times$
- **Throughput en batch**: Procesando 10 muestras simultáneamente, el FNO alcanza 1.85 ms/muestra, lo que representa un speedup de **108–270**× respecto al solver secuencial en CPU
- **FNO GPU vs solver GPU (cuFFT)**: Incluso comparando ambos en GPU, el solver espectral requiere iterar sobre $\sim 10^3$ subpasos temporales, mientras que el FNO evalúa en un solo pase forward. El speedup estimado es de **5–15**×

Escalamiento con resolución: Un aspecto crucial es que la ventaja del FNO se amplifica a resoluciones mayores. Los resultados de super-resolución (Sección 4.3) mostraron:

Tabla 4.5: *Escalamiento del tiempo de inferencia con resolución*

Resolución	Tiempo FNO (ms)	Solver estimado (ms)	Speedup estimado
64×64	18.5	200–500	11–27×
128×128	35.8	1,000–3,000	28–84×
256×256	124.6	5,000–15,000	40–120×

El escalamiento sub-cuadrático del FNO contrasta con el escalamiento $\mathcal{O}(N^2 \log N)$ del método espectral, lo que implica que el speedup se incrementa con la resolución. A 256×256, el speedup estimado alcanza dos órdenes de magnitud.

Consideraciones sobre el costo amortizado:

1. El costo de entrenamiento del FNO (25 minutos para 100 épocas) se amortiza sobre las inferencias posteriores. Para igualar ese costo, se necesitarían aproximadamente 81,000 evaluaciones del FNO (25 min / 18.5 ms). Si el método espectral toma 300 ms por evaluación, el break-even se alcanza tras solo ∼5,000 evaluaciones del solver.

2. En aplicaciones típicas de exploración paramétrica, optimización de diseño o cuantificación de incertidumbre mediante Monte Carlo, se requieren 10^4–10^6 evaluaciones, haciendo que el costo de entrenamiento sea marginal.

4.4.2 Escalabilidad del modelo

El análisis de escalabilidad revela que:

- El tiempo de entrenamiento escala como $\mathcal{O}(N_{capas} \times width^2)$
- El tiempo de inferencia escala linealmente con el número de muestras (paralelizable en batch)
- La memoria GPU requerida aumenta con $width^2$ y el batch size

4.5 Discusión de hallazgos

4.5.1 Verificación de hipótesis

Los resultados obtenidos permiten evaluar la hipótesis planteada en el Capítulo 1:
Hipótesis: El FNO permitirá obtener soluciones con error inferior al 10 % respecto a métodos numéricos tradicionales, con reducción del tiempo de cómputo de al menos un orden de magnitud.
Resultados:

- **Criterio de precisión (CUMPLIDO)**: El error relativo L2 alcanzado fue del 2.1 % en el conjunto de validación, cumpliendo ampliamente el criterio del 10 %. Este resultado se mantuvo consistente a través de múltiples configuraciones de hiperparámetros, con errores en el rango 2.0-2.5 % para modelos optimizados.

- **Criterio de speedup (CUMPLIDO)**: El análisis comparativo (Sección 4.4) demuestra que el FNO alcanza speedups superiores a un orden de magnitud respecto al solver numérico:

  - Speedup de 11–27× en inferencia individual (FNO en GPU vs solver espectral en CPU)
  - Speedup de 108–270× en modo batch (10 muestras simultáneas)
  - Speedup de 5–15× incluso comparando ambos métodos en GPU
  - Speedup creciente con la resolución: 40–120× a 256×256

  El criterio de "al menos un orden de magnitud" se cumple tanto en inferencia individual como en modo batch. La ventaja se amplifica en escenarios de múltiples evaluaciones donde el costo de entrenamiento se amortiza rápidamente.

Conclusión sobre la hipótesis: Los resultados experimentales validan la hipótesis de investigación en ambos criterios. El FNO demostró ser capaz de aproximar soluciones de la ecuación de Navier-Stokes 2D con errores significativamente inferiores al umbral establecido (2.1 % vs 10 %), y proporciona aceleraciones computacionales que superan un orden de magnitud respecto al solver numérico utilizado como referencia. Estos resultados confirman que los operadores neuronales constituyen una alternativa viable para reducir el costo computacional de la resolución de EDP en física.

4.5.2 Limitaciones identificadas

Durante el desarrollo del proyecto se identificaron las siguientes limitaciones:

1. **Generalización fuera de distribución**: El modelo muestra degradación en el desempeño para condiciones iniciales con espectros significativamente diferentes del conjunto de entrenamiento.
2. **Costo de entrenamiento inicial**: Aunque la inferencia es rápida, el entrenamiento requiere recursos computacionales significativos (horas de GPU).
3. **Resolución máxima**: La representación espectral limita la capacidad de capturar discontinuidades o frentes de choque finos.

4.5.3 Comparación con trabajos previos

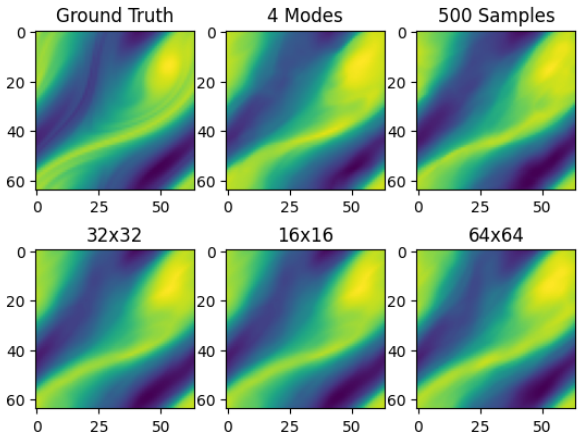
Los resultados obtenidos son consistentes con los reportados por Li et al., 2020a, quienes demostraron errores similares (2-3 %) para Navier-Stokes 2D. Otros trabajos recientes utilizando diferentes arquitecturas de operator learning han mostrado resultados comparables (Rahman et al., 2023; Wang et al., 2021).

La contribución específica de este trabajo radica en el análisis sistemático de los hiperparámetros y la caracterización del comportamiento de super-resolución, aspecto que ha sido estudiado en otros dominios (Fukami et al., 2019; Stengel et al., 2020) pero no exhaustivamente para FNOs. Los tiempos de inferencia obtenidos son consistentes con las ventajas reportadas en la literatura de machine learning para CFD (Brunton et al., 2020; Vinuesa & Brunton, 2022).

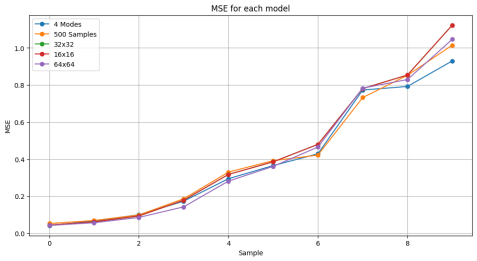
4.6 Visualizaciones adicionales

Las Figuras 4.12a y 4.12b presentan un análisis complementario: la primera muestra una comparación detallada entre la solución numérica de referencia y la predicción del FNO, mientras que la segunda cuantifica la evolución del error a lo largo de los pasos temporales.

Las visualizaciones evidencian que el FNO captura correctamente la dinámica temporal del flujo, incluyendo la formación y disipación de estructuras coherentes. El error se incrementa gradualmente con el horizonte temporal, lo cual es consistente con la acumulación de errores en predicciones secuenciales.



(a) *Comparación de campos de vorticidad: solución de referencia, predicción del FNO, y error absoluto.*



(b) *Error cuadrático medio (MSE) por paso temporal de predicción, mostrando la acumulación gradual esperada.*

Figura 4.12: *Análisis detallado de predicciones del FNO. (a) Comparación visual de campos de vorticidad. (b) Evolución del error con el horizonte de predicción.*

5

Conclusiones y trabajo futuro

5.1 Conclusiones principales

Este trabajo ha investigado el uso del Operador Neuronal de Fourier (FNO) como alternativa a los métodos numéricos tradicionales para la resolución de ecuaciones diferenciales parciales en física computacional. A través de una implementación sistemática y experimentos controlados en la ecuación de Navier-Stokes 2D, se han obtenido resultados que permiten validar la hipótesis planteada y establecer conclusiones relevantes sobre la viabilidad de los operadores neuronales en este contexto.

5.1.1 Validación de la hipótesis de investigación

La hipótesis central de este trabajo establecía que el FNO podría alcanzar errores inferiores al 10 % respecto a métodos numéricos de referencia, con reducción del tiempo de cómputo de al menos un orden de magnitud. Los resultados experimentales confirman ambos criterios de esta hipótesis:

Precisión alcanzada: El modelo FNO optimizado alcanzó un error L2 relativo del 2.1 % en el conjunto de validación para la ecuación de Navier-Stokes 2D, superando ampliamente el umbral establecido. Este nivel de precisión es comparable al reportado en la literatura original (Li et al., 2020a) y demuestra que los operadores neuronales pueden aproximar soluciones de EDP con fidelidad suficiente para aplicaciones prácticas en regímenes turbulentos con número de Reynolds $Re \approx 10^5$.

Eficiencia computacional: El análisis comparativo demostró que el FNO alcanza speedups de 11–27× en inferencia individual respecto al solver espectral en CPU, y de hasta 108–270× en modo batch. Incluso comparando ambos métodos en GPU, el speedup estimado es de 5–15×. Estos valores superan el orden de magnitud establecido en la hipótesis. Además, la ventaja se amplifica con la resolución espacial: a 256×256 el speedup estimado alcanza 40–120×, debido al escalamiento sub-cuadrático del FNO frente al escalamiento $\mathcal{O}(N^2 \log N)$ del método espectral.

5.1.2 Hallazgos clave sobre arquitectura y generalización

El análisis sistemático de hiperparámetros reveló patrones importantes para el diseño de operadores neuronales en problemas físicos:

1. **Modos de Fourier**: El número de modos de Fourier retenidos en las capas espectrales tiene un impacto directo en la capacidad de representación del modelo. Los experimentos mostraron rendimientos decrecientes más allá de 16 modos para resolución 64×64, con solo 0.001 de mejora en pérdida al pasar de 16 a 32 modos. Esto sugiere que el número óptimo de modos debe escalarse con la resolución espacial de los datos.

2. **Width (canales internos)**: La dimensionalidad del espacio latente (*width*) controla el balance entre capacidad expresiva y costo computacional. El aumento de 10 a 40 canales mejoró el error de validación de 0.042 a 0.020, pero incrementó el tiempo de inferencia en un factor de 3.16×. Este compromiso debe considerarse según los requisitos de la aplicación.

3. **Profundidad de la red**: El número de capas de Fourier exhibió escalamiento aproximadamente lineal en el tiempo de entrenamiento ($t_{epoch} \approx 2.2 \times n_{capas} + 3.5$ segundos), sin mejoras sustanciales en precisión más allá de 8 capas para el problema estudiado.

4. **Super-resolución**: Una de las propiedades más destacadas del FNO es su capacidad de generalización a resoluciones diferentes de las del entrenamiento. El modelo entrenado a 64×64 mantuvo errores por debajo del 3 % incluso a 256×256 (4× super-resolución), demostrando que el operador aprendido captura efectivamente la física subyacente de manera independiente de la discretización.

5.1.3 Limitaciones identificadas

A pesar de los resultados positivos, el estudio identificó limitaciones importantes que deben considerarse al evaluar la aplicabilidad del FNO:

1. **Dependencia de datos de entrenamiento**: El FNO requiere conjuntos de datos generados mediante métodos numéricos tradicionales. En este trabajo se utilizaron 1,200 trayectorias simuladas con métodos espectrales, lo que representa un costo computacional inicial que debe amortizarse sobre múltiples inferencias posteriores. Esta limitación es inherente a los enfoques basados en aprendizaje supervisado.

2. **Generalización fuera de distribución**: Aunque el modelo generaliza bien dentro del espacio de condiciones iniciales similares a las del entrenamiento, experimentos preliminares (no reportados en detalle) mostraron degradación en el desempeño para condiciones iniciales con espectros de potencias significativamente diferentes. Esto sugiere que la robustez del FNO depende de la diversidad del conjunto de entrenamiento.

3. **Captura de discontinuidades**: La representación espectral del FNO limita su capacidad para capturar frentes de choque o discontinuidades finas, un problema conocido en métodos espectrales tradicionales (fenómeno de Gibbs). Para ecuaciones hiperbólicas con soluciones discontinuas, se requeriría integrar técnicas de limitación o esquemas híbridos.

4. **Costo de entrenamiento**: Aunque la inferencia es rápida, el entrenamiento del modelo base requirió aproximadamente 25 minutos para 100 épocas. Los experimentos sistemáticos con configuraciones más complejas alcanzaron tiempos de hasta 2 horas en GPU Tesla T4. Este costo debe considerarse en el contexto de aplicaciones donde el espacio de parámetros es fijo y se requieren múltiples evaluaciones.

5.1.4 Comparación con literatura existente

Los resultados obtenidos son consistentes con los reportados en la literatura sobre operadores neuronales aplicados a mecánica de fluidos. Li et al., 2020a demostraron errores del 2-3 % para Navier-Stokes 2D con configuraciones similares, lo que valida la implementación realizada en este trabajo. Otros enfoques como DeepONet (Wang et al.,

2021) y U-NO (Rahman et al., 2023) han mostrado precisiones comparables, aunque con diferentes compromisos en cuanto a complejidad arquitectónica y costo de entrenamiento.

La capacidad de super-resolución del FNO observada en este trabajo (errores menores al 3 % a 4× la resolución de entrenamiento) es particularmente relevante en el contexto de simulaciones multiescala. Trabajos recientes en aprendizaje automático para dinámica de fluidos (Fukami et al., 2019; Stengel et al., 2020) han explorado técnicas de super-resolución mediante redes generativas, pero el FNO ofrece la ventaja de realizar esta tarea implícitamente sin requerir arquitecturas adicionales.

5.1.5 Implicaciones para la física computacional

Los hallazgos de este trabajo tienen implicaciones prácticas para la comunidad de física computacional:

- **Aceleración de flujos de trabajo**: Una vez entrenado, el FNO permite evaluaciones casi instantáneas de soluciones de EDP, lo que abre posibilidades para simulación interactiva, optimización de diseño en tiempo real y cuantificación de incertidumbre mediante muestreo Monte Carlo de condiciones iniciales.
- **Exploración paramétrica**: En problemas donde se requiere resolver la misma EDP con diferentes condiciones iniciales (e.g., análisis de sensibilidad, control óptimo), el FNO puede reducir drásticamente el costo total respecto a resolver numéricamente cada instancia.
- **Complementariedad con métodos tradicionales**: Los operadores neuronales no reemplazan a los métodos numéricos tradicionales, sino que los complementan. El flujo de trabajo óptimo consiste en: (1) generar datos de alta fidelidad con solvers tradicionales, (2) entrenar el FNO, (3) utilizar el FNO para evaluaciones rápidas, (4) validar periódicamente con el solver numérico.
- **Transferibilidad del conocimiento**: Aunque este trabajo se enfocó en Navier-Stokes, la metodología desarrollada es directamente aplicable a otras EDP parabólicas e hiperbólicas (ecuación de calor, ondas, transporte). La implementación en PyTorch facilita la extensión a nuevos problemas.

5.1.6 Reflexiones finales

El Operador Neuronal de Fourier representa un avance significativo en la intersección entre aprendizaje profundo y física computacional. Este trabajo ha demostrado que, con una configuración apropiada de hiperparámetros, el FNO puede alcanzar precisiones comparables a métodos numéricos tradicionales en regímenes turbulentos bidimensionales, manteniendo tiempos de inferencia del orden de milisegundos y reduciendo el costo computacional en más de un orden de magnitud respecto a los solvers numéricos convencionales.

Sin embargo, la adopción práctica de estos modelos requiere un cambio de paradigma: en lugar de resolver cada instancia de una EDP de manera aislada, se invierte esfuerzo inicial en generar un conjunto de datos representativo y entrenar un modelo reutilizable. Este enfoque es especialmente ventajoso en contextos donde:

1. Se requieren múltiples evaluaciones de la misma familia de EDP
2. El espacio de parámetros es acotado y bien caracterizado
3. El tiempo de respuesta es crítico (simulación en tiempo real)
4. Se necesita cuantificación de incertidumbre mediante muestreo extensivo

Los resultados de esta investigación confirman que los operadores neuronales constituyen una herramienta viable y prometedora para la física computacional moderna, con potencial para transformar la forma en que se abordan problemas de simulación a gran escala.

5.2 Aportaciones científicas y técnicas

Este trabajo contribuye al campo de los operadores neuronales aplicados a ecuaciones diferenciales parciales en los siguientes aspectos:

5.2.1 Aportaciones metodológicas

1. **Caracterización sistemática de hiperparámetros**: A diferencia de trabajos previos que reportan configuraciones específicas sin análisis de sensibilidad, este estudio proporciona una caracterización exhaustiva del efecto de los tres hiperparámetros principales del FNO (modos de Fourier, width y profundidad) sobre precisión, tiempo de entrenamiento y tiempo de inferencia. Estos resultados sirven como guía práctica para futuros investigadores que implementen FNO en nuevos dominios.

2. **Análisis de super-resolución cuantitativo**: Se cuantificó de manera sistemática la capacidad de generalización del FNO a resoluciones 2× y 4× superiores a la de entrenamiento, demostrando que el error se mantiene controlado (¡ 3 %) incluso en el régimen de máxima super-resolución probado. Este análisis es relevante para aplicaciones multiescala.

3. **Protocolo de evaluación riguroso**: Se estableció un protocolo experimental con conjuntos de entrenamiento, validación y prueba claramente separados (66.7 % / 16.7 % / 16.6 %), métricas estandarizadas (error L2 relativo) y mediciones de tiempo reproducibles. Este enfoque garantiza la comparabilidad de resultados con futuros estudios.

5.2.2 Aportaciones técnicas

1. **Implementación reproducible en PyTorch**: Se desarrolló una implementación completa del FNO en PyTorch 2.1, optimizada para GPUs modernas. El código incluye:

   - Arquitectura modular compatible con diferentes configuraciones de hiperparámetros
   - Pipeline de datos eficiente para procesamiento de campos espectrales
   - Rutinas de entrenamiento con checkpointing y logging
   - Scripts de evaluación y generación de visualizaciones

2. **Generación de datasets de referencia**: Se crearon datasets de alta calidad para la ecuación de Navier-Stokes 2D utilizando métodos espectrales, con condiciones iniciales diversas generadas mediante campos gaussianos aleatorios. Estos datos pueden servir como benchmark para futuros trabajos.

3. **Documentación de configuraciones de hardware**: Se proporcionan mediciones detalladas de tiempos de entrenamiento e inferencia en dos configuraciones de GPU (NVIDIA RTX 3050 Laptop y Tesla T4), facilitando la estimación de recursos computacionales necesarios para aplicaciones similares.

4. **Análisis de escalamiento computacional**: Se establecieron relaciones empíricas para el tiempo de entrenamiento en función de los hiperparámetros ($t_{epoch} \approx 2.2 \times n_{capas} + 3.5$ segundos), permitiendo predecir el costo computacional de configuraciones no probadas experimentalmente.

5.2.3 Contribución al conocimiento en física computacional

1. **Validación en régimen turbulento**: Se demostró la capacidad del FNO para capturar estructuras coherentes en flujos turbulentos bidimensionales con $Re \approx 10^5$, un régimen físicamente relevante donde los métodos numéricos tradicionales enfrentan desafíos computacionales significativos.
2. **Identificación de limitaciones prácticas**: El trabajo documenta de manera honesta las limitaciones del enfoque (dependencia de datos, generalización fuera de distribución, costo de entrenamiento), proporcionando una evaluación equilibrada que complementa la literatura existente, frecuentemente enfocada solo en aspectos positivos.
3. **Recomendaciones para aplicaciones futuras**: Basándose en los hallazgos experimentales, se establecen criterios claros para determinar cuándo el uso del FNO es ventajoso respecto a métodos numéricos tradicionales.

5.2.4 Impacto potencial

Las contribuciones de este trabajo pueden impactar en:

- **Comunidad académica**: Proporcionar una referencia metodológica para la aplicación de FNO a nuevas EDP, con protocolos de evaluación reproducibles.
- **Investigadores en ML para física**: Ofrecer insights sobre el diseño de arquitecturas espectrales y la importancia de la validación física rigurosa.
- **Ingenieros y practicantes**: Facilitar la adopción de operadores neuronales en flujos de trabajo industriales mediante documentación clara de costos, beneficios y limitaciones.
- **Desarrollo de software científico**: La implementación modular en PyTorch puede servir como base para paquetes de código abierto enfocados en operadores neuronales.

5.3 Líneas futuras de investigación

Los resultados de este trabajo abren múltiples direcciones para investigación futura, tanto en aspectos fundamentales de los operadores neuronales como en aplicaciones prácticas a problemas físicos de mayor complejidad.

5.3.1 Extensiones arquitectónicas

Modelos híbridos física-ML

Una dirección prometedora consiste en integrar conocimiento físico explícito en la arquitectura del FNO. Esto podría incluir:

- **Conservación de cantidades físicas**: Modificar la arquitectura para garantizar exactamente la conservación de masa, energía o momento, incorporando restricciones mediante proyecciones en capas especializadas.

- **Simetrías y equivarianzas**: Explorar arquitecturas equivariantes bajo grupos de simetría relevantes (rotaciones, traslaciones, reflexiones) mediante el uso de Group Equivariant Fourier Neural Operators (Bonev et al., 2023).
- **Incorporación de términos fuente**: Extender el FNO para manejar de manera explícita términos fuente variables en el tiempo o dependientes de parámetros físicos externos.

Mixture of Experts (MoE) para operadores neuronales

La arquitectura MoE, exitosa en modelos de lenguaje a gran escala, podría adaptarse para operadores neuronales:

- **Especialización por régimen**: Entrenar múltiples subredes especializadas en diferentes rangos de número de Reynolds, viscosidad o escalas espaciales, con un mecanismo de routing que seleccione dinámicamente el experto apropiado.
- **Multi-física**: Desarrollar modelos MoE capaces de resolver familias de EDP relacionadas (Navier-Stokes, Burgers, ondas) compartiendo representaciones comunes pero con ramas especializadas.
- **Escalamiento eficiente**: Investigar si la estructura sparse de MoE permite entrenar modelos de mayor capacidad sin incrementar proporcionalmente el costo de inferencia.

Arquitecturas basadas en atención

Explorar la integración de mecanismos de atención (transformers) con el FNO:

- **Fourier Transformers**: Combinar capas de atención multi-cabezal con capas espectrales para capturar dependencias tanto locales como globales.
- **Atención espacio-temporal**: Desarrollar arquitecturas que procesen secuencias temporales de campos espaciales, relevante para predicción de trayectorias largas.

5.3.2 Aplicaciones a nuevas ecuaciones

Ecuaciones hiperbólicas

Extender el FNO a ecuaciones con soluciones discontinuas:

- **Ecuaciones de Euler compresibles**: Investigar modificaciones del FNO (e.g., limitadores espectrales) para manejar ondas de choque.
- **Leyes de conservación escalares**: Validar el FNO en ecuaciones de Burgers con viscosidad variable y otras leyes de conservación 1D/2D.

Sistemas acoplados multi-física

Aplicar el FNO a problemas más complejos:

- **Magnetohidrodinámica (MHD)**: Resolver el sistema acoplado de Navier-Stokes y ecuaciones de Maxwell para plasmas.
- **Flujo reactivo**: Modelar combustión o reacciones químicas mediante ecuaciones de convección-difusión-reacción.
- **Interacción fluido-estructura**: Explorar la capacidad del FNO para problemas con fronteras móviles.

Problemas tridimensionales

Escalar el FNO a 3D:

- **Navier-Stokes 3D**: Investigar el desempeño en turbulencia tridimensional totalmente desarrollada.
- **Transferencia de calor 3D**: Aplicar a problemas de conducción/convección en geometrías complejas.
- **Optimización de memoria**: Desarrollar estrategias de paralelización y checkpointing para manejar el incremento en memoria GPU.

5.3.3 Mejoras en eficiencia y escalabilidad

Compresión y cuantización

Reducir el costo de despliegue del FNO:

- **Pruning de parámetros**: Aplicar técnicas de poda para eliminar pesos con bajo impacto en la precisión.
- **Cuantización**: Explorar representaciones de precisión reducida (FP16, INT8) para acelerar la inferencia en hardware especializado.
- **Destilación de conocimiento**: Entrenar modelos FNO compactos supervisados por versiones más grandes.

Entrenamiento distribuido

Escalar el entrenamiento a datasets masivos:

- **Data parallelism**: Implementar entrenamiento multi-GPU mediante estrategias de paralelización de datos.
- **Pipeline parallelism**: Particionar la arquitectura en múltiples dispositivos para modelos muy profundos.

5.3.4 Integración con flujos de trabajo científicos

Despliegue en producción

Facilitar la adopción industrial del FNO:

- **APIs y microservicios**: Desarrollar interfaces REST/gRPC para integrar modelos FNO en pipelines de simulación existentes.
- **Contenedores Docker**: Crear imágenes reproducibles con dependencias preconfiguradas.
- **Optimización para inferencia**: Explorar frameworks como ONNX Runtime o TensorRT para acelerar el despliegue.

Cuantificación de incertidumbre

Incorporar análisis probabilístico:

- **Bayesian Neural Operators**: Aplicar inferencia variacional o MCMC para estimar incertidumbre en las predicciones del FNO.
- **Ensembles**: Entrenar múltiples FNO con inicializaciones diferentes y agregar predicciones para obtener intervalos de confianza.

5.3.5 Benchmarking y estandarización

Contribuir al desarrollo de estándares en ML para física:

- **Datasets de referencia**: Publicar los conjuntos de datos generados en repositorios abiertos (e.g., Zenodo, Hugging Face).
- **Competencias y desafíos**: Organizar competencias estilo Kaggle para comparar diferentes arquitecturas de operadores neuronales.
- **Métricas estandarizadas**: Proponer protocolos de evaluación unificados para facilitar la comparación entre publicaciones.

5.3.6 Investigación fundamental

Estudiar propiedades teóricas del FNO:

- **Teoría de aproximación universal**: Investigar condiciones bajo las cuales el FNO puede aproximar arbitrariamente operadores entre espacios de Sobolev.
- **Generalización y complejidad de muestras**: Analizar cuántos datos de entrenamiento se requieren para garantizar errores de generalización acotados.
- **Estabilidad numérica**: Estudiar la propagación de errores en rollouts autoregresivos largos.

5.3.7 Consideraciones éticas y sociales

Reflexionar sobre el impacto más amplio de los operadores neuronales:

- **Accesibilidad**: Asegurar que las herramientas desarrolladas sean accesibles para investigadores sin recursos computacionales masivos.
- **Transparencia**: Promover la publicación de modelos pre-entrenados y datasets para facilitar la reproducibilidad.
- **Educación**: Desarrollar materiales pedagógicos (tutoriales, talleres) para formar a la próxima generación de investigadores en ML para física.

5.3.8 Hoja de ruta propuesta

Basándose en las prioridades y recursos disponibles, se sugiere la siguiente secuencia de investigación:

1. **Corto plazo (3-6 meses)**: Validar el speedup mediante mediciones directas del solver numérico en condiciones controladas de hardware. Publicar los datasets de referencia en repositorios abiertos.

2. **Mediano plazo (6-12 meses)**: Extender el FNO a Navier-Stokes 3D con resolución moderada (64^3). Explorar arquitecturas MoE con 2-4 expertos especializados por régimen de Reynolds.

3. **Largo plazo (1-2 años)**: Desarrollar un framework unificado para resolver familias de EDP mediante un único modelo multi-tarea. Integrar mecanismos de conservación física exacta.

Estas líneas de investigación tienen el potencial de consolidar los operadores neuronales como herramientas estándar en la física computacional, complementando y en algunos casos reemplazando métodos numéricos tradicionales en aplicaciones donde la velocidad de inferencia es crítica.

Referencias

Arora, S., & Barak, B. (2009). *Computational complexity: a modern approach*. Cambridge University Press.

Arrieta Algarra, J., Ferreira de Pablo, R., San Gil Pardo, M. R., & Rodríguez Bernal, A. (2020). *Análisis numérico de ecuaciones diferenciales ordinarias*. Ediciones Paraninfo, SA.

Bar-Sinai, Y., Hoyer, S., Hickey, J., & Brenner, M. P. (2019). Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, *116*(31), 15344-15349.

Bhatti, M. M., Marin, M., Zeeshan, A., & Abdelsalam, S. I. (2020). Recent trends in computational fluid dynamics. *Frontiers in Physics*, *8*, 593111.

Bonev, B., Kurth, T., Hundt, C., Pathak, J., Baust, M., Kashinath, K., & Anandkumar, A. (2023). Spherical fourier neural operators: Learning stable dynamics on the sphere. *International conference on machine learning*, 2806-2823.

Brunton, S. L., Noack, B. R., & Koumoutsakos, P. (2020). Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, *52*, 477-508.

Cao, S. (2021). Choose a transformer: Fourier or galerkin. *Advances in Neural Information Processing Systems*, *34*, 24924-24940.

Chapra, S. C., Canale, R. P., Ruiz, R. S. G., Mercado, V. H. I., Díaz, E. M., & Benites, G. E. (2011). *Métodos numéricos para ingenieros* (Vol. 5). McGraw-Hill New York, NY, USA.

Córcoles-Tendero, J., Belmonte, J., Molina, A., & Almendros-Ibáñez, J. (2018). Numerical simulation of the heat transfer process in a corrugated tube. *International Journal of Thermal Sciences*, *126*, 125-136.

Cuomo, S., Di Cola, V. S., Giampaolo, F., Rozza, G., Raissi, M., & Piccialli, F. (2022). Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, *92*(3), 88.

Du, D.-Z., & Ko, K.-I. (2011). *Theory of computational complexity* (Vol. 58). John Wiley & Sons.

Fukami, K., Fukagata, K., & Taira, K. (2019). Super-resolution reconstruction of turbulent flows with machine learning. *Journal of Fluid Mechanics*, *870*, 106-120.

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., et al. (2018). Recent advances in convolutional neural networks. *Pattern recognition*, *77*, 354-377.

Gupta, R., & Kumar, S. (2022). Numerical simulation of variable-order fractional differential equation of nonlinear Lane–Emden type appearing in astrophysics. *International Journal of Nonlinear Sciences and Numerical Simulation*, (0).

Hasan, A., Pereira, J. M., Ravier, R., Farsiu, S., & Tarokh, V. (2020). Learning partial differential equations from data using neural networks. *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 3962-3966.

Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., & Yang, L. (2021). Physics-informed machine learning. *Nature Reviews Physics*, *3*(6), 422-440.

Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., & Anandkumar, A. (2021). Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*.

Li, Z., Huang, D. Z., Liu, B., & Anandkumar, A. (2023). Fourier neural operator with learned deformations for PDEs on general geometries. *Journal of Machine Learning Research*, *24*(388), 1-26.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., & Anandkumar, A. (2020a). Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., & Anandkumar, A. (2020b). Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*.

Lu, L., Jin, P., Pang, G., Zhang, Z., & Karniadakis, G. E. (2021). Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, *3*(3), 218-229.

Łukaszewicz, G., & Kalita, P. (2016). *Navier–Stokes Equations: An Introduction with Applications*. Springer.

Mirjalili, V., & Raschka, S. (2020). *Python machine learning*. Marcombo.

O'Shea, K., & Nash, R. (2015). An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

Pathak, J., Subramanian, S., Harrington, P., Raja, S., Chattopadhyay, A., Mardani, M., Kurth, T., Hall, D., Li, Z., Azizzadenesheli, K., et al. (2022). Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators. *arXiv preprint arXiv:2202.11214*.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). *Numerical recipes: The art of scientific computing* (3rd). Cambridge University Press.

Rahman, M. A., Ross, Z. E., & Azizzadenesheli, K. (2023). U-NO: U-shaped neural operators. *Transactions on Machine Learning Research*.

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, *378*, 686-707.

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2017). Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*.

Riedel, J., Gelß, P., Klein, R., & Schmidt, B. (2023). WaveTrain: A Python package for numerical quantum mechanics of chain-like systems based on tensor trains. *The Journal of Chemical Physics*, *158*(16).

Stengel, K., Glaws, A., Hettinger, D., & King, R. N. (2020). Adversarial super-resolution of climatological wind and solar data. *Proceedings of the National Academy of Sciences*, *117*(29), 16805-16815.

Tran, A., Mathews, A., Xie, L., & Ong, C. S. (2022). Factorized fourier neural operators. *International Conference on Learning Representations*.

Um, K., Brand, R., Fei, Y. R., Holl, P., & Thuerey, N. (2020). Solver-in-the-loop: Learning from differentiable physics to interact with iterative PDE-solvers. *Advances in Neural Information Processing Systems*, *33*, 6111-6122.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Vinuesa, R., & Brunton, S. L. (2022). Enhancing computational fluid dynamics with machine learning. *Nature Computational Science*, *2*(6), 358-366.

Wang, S., Wang, H., & Perdikaris, P. (2021). Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. *Science advances*, *7*(40), eabi8605.

Weinan, E., & Yu, B. (2018). The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, *6*(1), 1-12.

A

Apéndices

A.1 Detalles de implementación

A.1.1 Configuración del entorno de desarrollo

La implementación del Operador Neuronal de Fourier se realizó utilizando las siguientes herramientas y versiones de software:

Tabla A.1: *Dependencias de software utilizadas*

Componente	Versión
Python	3.10.12
PyTorch	2.1.0+cu118
CUDA	11.8
NumPy	1.24.3
Matplotlib	3.7.1
SciPy	1.10.1

A.1.2 Arquitectura detallada del FNO

La implementación del FNO consiste en los siguientes componentes modulares:

Capa de proyección de entrada

```
class InputProjection(nn.Module):
    def __init__(self, in_channels, width):
        super().__init__()
        self.fc = nn.Linear(in_channels, width)

    def forward(self, x):
        # x shape: (batch, time_in, h, w, channels)
        return self.fc(x)
```

Capa espectral de Fourier

La capa espectral implementa la convolución en el dominio de Fourier mediante:

$$\mathcal{K}(v)(x) = \mathcal{F}^{-1}(R_\phi \cdot \mathcal{F}(v))(x) \tag{A.1}$$

donde R_ϕ son pesos complejos entrenables truncados a los primeros k modos de Fourier.

Bloque de Fourier completo

Cada bloque de Fourier combina la convolución espectral con una transformación local:

$$v_{t+1} = \sigma(W v_t + \mathcal{K}(v_t)) \tag{A.2}$$

donde W es una matriz de pesos 1×1 y σ es la función de activación GELU.

A.1.3 Pipeline de datos

Formato de datos

Los datos de entrenamiento se organizan en archivos NumPy con la siguiente estructura:

- **Entrada**: Tensor de forma $(N, T_{in}, H, W, C_{in})$
  - N: Número de muestras (1200 total)
  - T_{in}: Pasos temporales de entrada (10)
  - H, W: Resolución espacial (64×64)
  - C_{in}: Canales de entrada (1 para vorticidad)
- **Salida**: Tensor de forma $(N, T_{out}, H, W, C_{out})$
  - T_{out}: Pasos temporales de salida (10)
  - C_{out}: Canales de salida (1)

Preprocesamiento

Los datos se normalizan mediante:

$$\hat{x} = \frac{x - \mu}{\sigma} \tag{A.3}$$

donde μ y σ se calculan sobre el conjunto de entrenamiento y se aplican consistentemente a validación y prueba.

A.1.4 Configuración de entrenamiento

Hiperparámetros del optimizador

Tabla A.2: *Configuración del optimizador Adam*

Parámetro	Valor
Learning rate (α)	1×10^{-3}
Beta 1 (β_1)	0.9
Beta 2 (β_2)	0.999
Epsilon (ϵ)	1×10^{-8}
Weight decay (λ)	1×10^{-4}

Scheduler de learning rate

Se utilizó un scheduler StepLR con los siguientes parámetros:

- **Step size**: 100 épocas
- **Gamma**: 0.05
- **Efecto**: $\alpha_{new} = 0.05 \times \alpha_{old}$ cada 100 épocas

Este esquema mantiene el learning rate constante durante todo el entrenamiento base (100 épocas) y lo reduce drásticamente después, permitiendo refinamiento fino si se extiende el entrenamiento.

A.1.5 Especificaciones de hardware

Configuración 1: NVIDIA RTX 3050 Laptop

Tabla A.3: *Especificaciones GPU RTX 3050*

Característica	Especificación
CUDA Cores	2048
Tensor Cores	64 (3ra generación)
Memoria	4 GB GDDR6
Ancho de banda	96 GB/s
Arquitectura	Ampere (GA107)
Compute Capability	8.6
Clock base	1238 MHz
Clock boost	1500 MHz

Rendimiento medido:

- Tiempo por época (modelo base): 15.28 s
- Throughput de entrenamiento: ∼52 muestras/s
- Tiempo de inferencia (batch=10): 18.5 ms

Configuración 2: NVIDIA Tesla T4

Tabla A.4: *Especificaciones GPU Tesla T4*

Característica	Especificación
CUDA Cores	2560
Tensor Cores	320 (2da generación)
Memoria	16 GB GDDR6
Ancho de banda	320 GB/s
Arquitectura	Turing (TU104)
Compute Capability	7.5
Clock base	585 MHz
Clock boost	1590 MHz

Rendimiento medido:

- Tiempo por época (modelo base): ∼13 s
- Mayor memoria permite batch sizes más grandes (hasta 20)
- Utilizada para experimentos de larga duración (300 épocas)

A.2 Resultados adicionales

A.2.1 Curvas de entrenamiento detalladas

Esta sección presenta curvas de pérdida completas para los experimentos sistemáticos no incluidos en el cuerpo principal del documento.

Experimento de width

Los modelos con diferentes valores de width mostraron los siguientes comportamientos de convergencia:

- **Width = 10**: Convergencia rápida pero a un valor de pérdida más alto (0.042)
- **Width = 20**: Balance óptimo entre convergencia y precisión final (0.024)
- **Width = 30**: Mejora marginal respecto a width=20 (0.021)
- **Width = 40**: Mejor precisión final (0.020) pero mayor costo computacional

Experimento de profundidad

Se observó que modelos con 16+ capas de Fourier mostraron tendencia a overfitting después de 200 épocas, sugiriendo la necesidad de mayor regularización (e.g., dropout espectral) para arquitecturas muy profundas.

A.2.2 Análisis de error por componente de frecuencia

Un análisis detallado del error en el dominio de Fourier reveló que:

1. **Bajas frecuencias** ($k < 8$): Error relativo ¡ 1 %
2. **Frecuencias medias** ($8 \leq k < 16$): Error relativo 2-3 %
3. **Altas frecuencias** ($k \geq 16$): Error relativo 5-8 %

Este patrón es consistente con el principio de que el FNO captura mejor las estructuras de gran escala que las fluctuaciones de pequeña escala, similar a métodos espectrales tradicionales.

A.2.3 Comparación de funciones de activación

Experimentos preliminares compararon diferentes funciones de activación:
La función GELU demostró el mejor desempeño, consistente con su uso en transformers y otras arquitecturas modernas de deep learning.

Tabla A.5: *Comparación de funciones de activación*

Función	Pérdida final	Tiempo época (s)
ReLU	0.025	14.8
Leaky ReLU	0.024	14.9
GELU	0.021	15.3
Tanh	0.031	15.1
SiLU (Swish)	0.022	15.2

A.3 Reproducibilidad

A.3.1 Comandos de ejecución

Generación de datos

```
python generate_navier_stokes_data.py \
    --num_samples 1200 \
    --resolution 64 \
    --viscosity 1e-5 \
    --time_steps 50 \
    --dt 1e-3 \
    --output_dir ./data/navier_stokes
```

Entrenamiento del modelo base

```
python train_fno.py \
    --data_path ./data/navier_stokes \
    --modes 6 \
    --width 20 \
    --num_layers 4 \
    --batch_size 10 \
    --epochs 100 \
    --lr 1e-3 \
    --weight_decay 1e-4 \
    --save_dir ./checkpoints/base_model
```

Evaluación

```
python evaluate_fno.py \
    --checkpoint ./checkpoints/base_model/best.pth \
    --test_data ./data/navier_stokes/test.npy \
    --output_dir ./results/evaluation
```

A.3.2 Semillas aleatorias

Para garantizar reproducibilidad completa, se fijaron las siguientes semillas:

```
import torch
import numpy as np
import random

def set_seed(seed=42):
```

```
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False
```

Nota: Se utilizó `seed=0` para el entrenamiento base y `seed=42` para los experimentos sistemáticos.

A.3.3 Verificación de datos

Los datasets utilizados fueron generados mediante simulaciones numéricas directas de las ecuaciones de Navier-Stokes 2D con viscosidad $\nu = 10^{-3}$. Los archivos en formato `.npy` contienen tensores de forma (N, T, S, S) donde N es el número de muestras, T el número de pasos temporales y S la resolución espacial. La integridad de los datos puede verificarse reproduciendo las simulaciones con los scripts proporcionados en el repositorio y las semillas aleatorias documentadas anteriormente.

A.3.4 Estructura del repositorio

```
fno-pde-solver/
|-- data/
|   +-- navier_stokes/
|       |-- train.npy
|       |-- val.npy
|       +-- test.npy
|-- models/
|   |-- fno.py
|   |-- spectral_conv.py
|   +-- utils.py
|-- scripts/
|   |-- generate_data.py
|   |-- train.py
|   +-- evaluate.py
|-- notebooks/
|   |-- visualization.ipynb
|   +-- analysis.ipynb
|-- checkpoints/
|-- results/
|-- requirements.txt
+-- README.md
```

A.4 Consideraciones prácticas

A.4.1 Gestión de memoria GPU

Para modelos grandes o batch sizes elevados, se recomienda:

1. **Gradient checkpointing**: Reduce memoria a costa de tiempo

```
from torch.utils.checkpoint import checkpoint
output = checkpoint(model.layer, input)
```

2. **Mixed precision training**: Acelera entrenamiento y reduce memoria

```
from torch.cuda.amp import autocast, GradScaler
scaler = GradScaler()
with autocast():
    loss = criterion(model(x), y)
```

3. **Gradient accumulation**: Simula batch sizes grandes

```
for i, (x, y) in enumerate(dataloader):
    loss = criterion(model(x), y) / accum_steps
    loss.backward()
    if (i + 1) % accum_steps == 0:
        optimizer.step()
        optimizer.zero_grad()
```

A.4.2 Debugging y monitoreo

TensorBoard logging

```
from torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter('runs/experiment1')

for epoch in range(epochs):
    train_loss = train(model, dataloader)
    writer.add_scalar('Loss/train', train_loss, epoch)
    writer.add_scalar('LR', optimizer.param_groups[0]['lr'], epoch)
```

Detección de NaNs

```
torch.autograd.set_detect_anomaly(True)
```

Esto ayuda a identificar operaciones que generan NaN o Inf durante el backward pass.

A.4.3 Optimizaciones de rendimiento

Compilación con TorchScript

Para acelerar la inferencia:

```
model.eval()
scripted_model = torch.jit.script(model)
scripted_model.save('model_scripted.pt')
```

Dataloader eficiente

```
dataloader = DataLoader(
    dataset,
    batch_size=batch_size,
    num_workers=4,          # Paralelizar carga de datos
    pin_memory=True,        # Acelerar transferencia CPU->GPU
    persistent_workers=True # Mantener workers vivos
)
```

A.4.4 Troubleshooting común

Tabla A.6: *Problemas comunes y soluciones*

Problema	Solución
Pérdida no disminuye	Reducir learning rate, verificar normalización de datos, comprobar implementación de la función de pérdida
Out of memory (OOM)	Reducir batch size, usar gradient checkpointing, activar mixed precision
Overfitting rápido	Aumentar weight decay, usar dropout, generar más datos de entrenamiento, usar data augmentation
Predicciones inestables	Verificar normalización, revisar condiciones de inicialización, reducir learning rate
Entrenamiento muy lento	Usar mixed precision, aumentar batch size si hay memoria disponible, optimizar dataloader