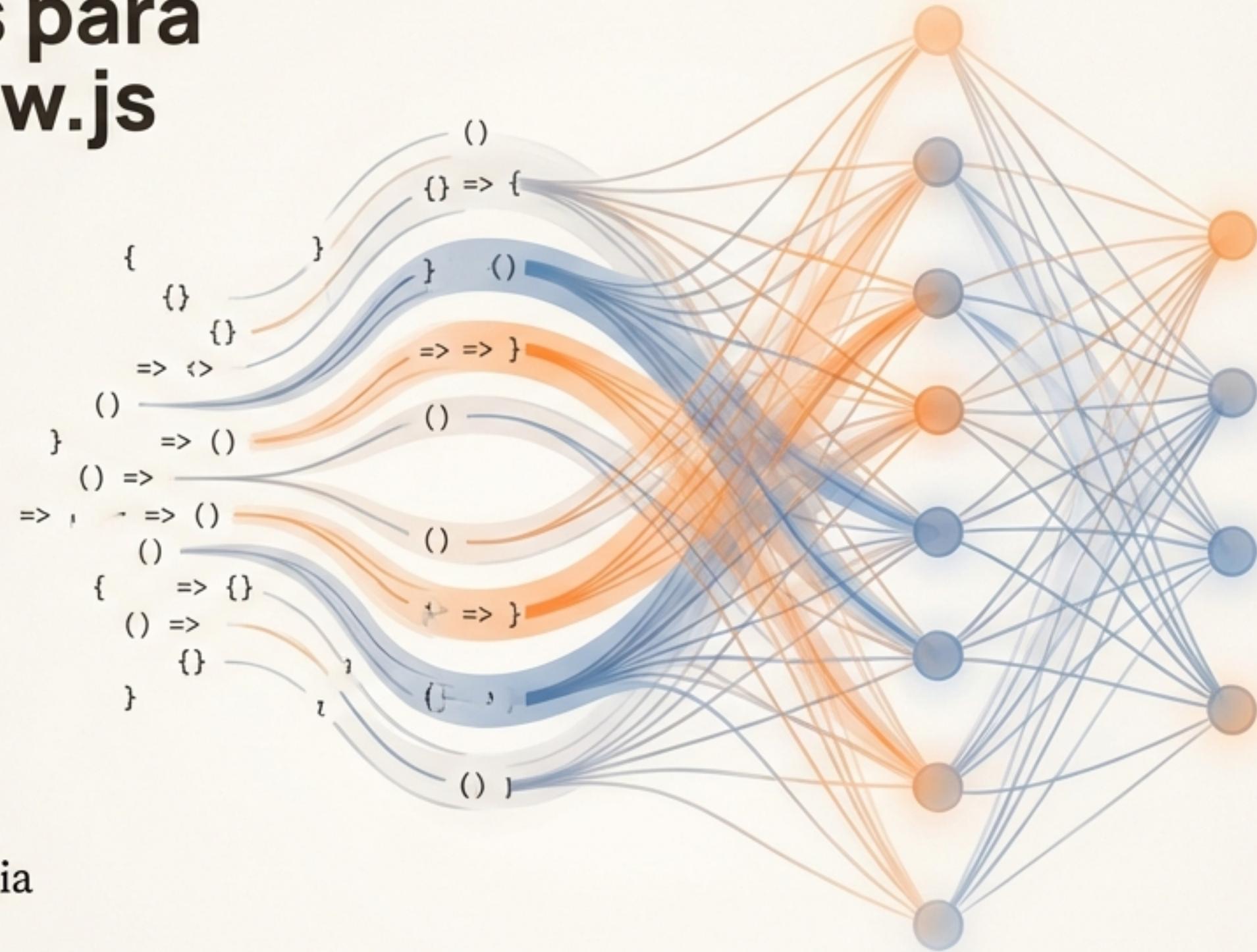


IA para Desarrolladores

JavaScript: Superpoderes para la Web con TensorFlow.js

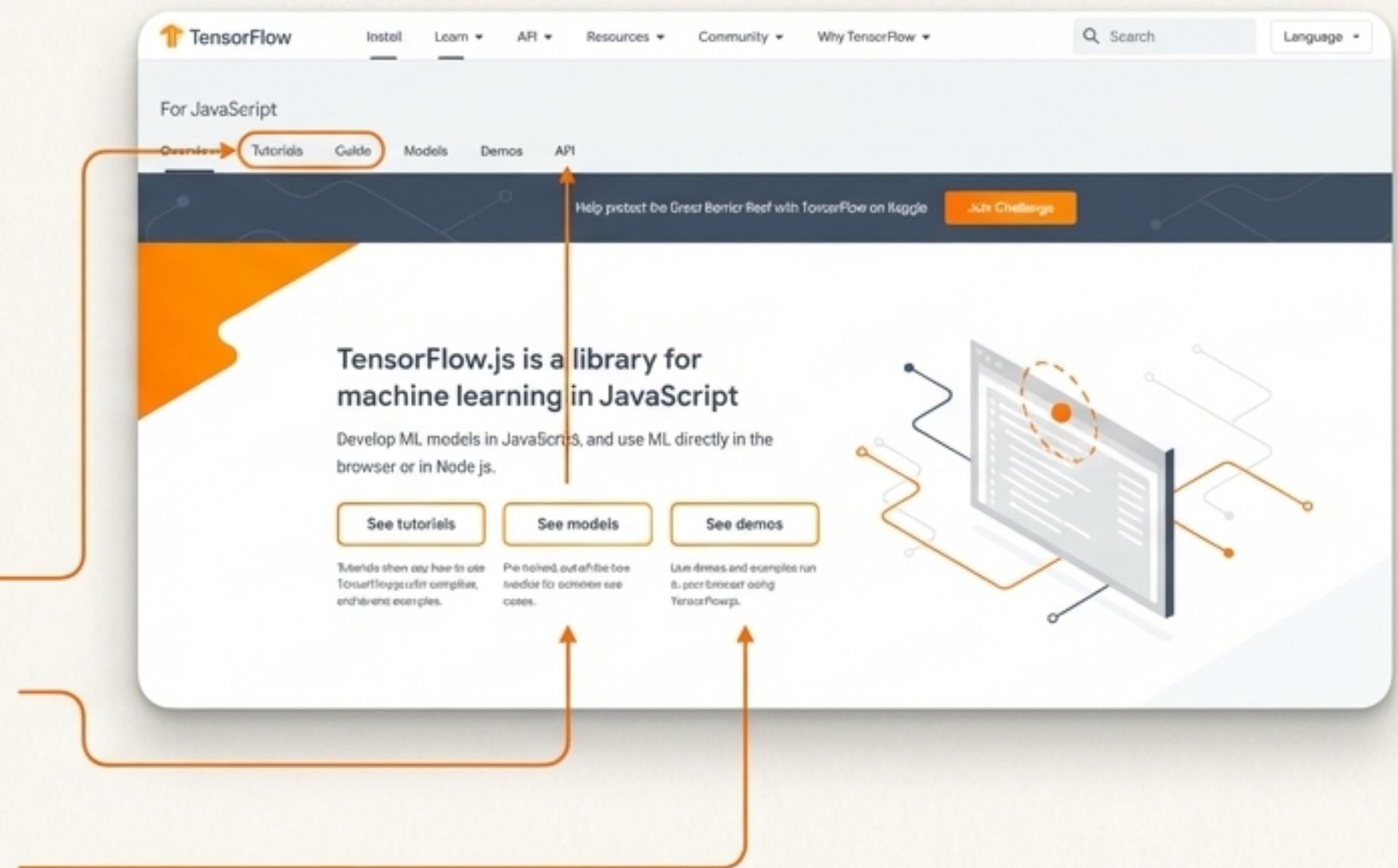
- **Visión:** Supercargar sitios y aplicaciones web para que actúen de forma inteligente.
- ⚙️ **Definición:** La biblioteca de Machine Learning de Google para el ecosistema JavaScript y Node.js.
- 👤 **Público Objetivo:** Diseñado para desmitificar la IA para desarrolladores JS, sin necesidad de experiencia previa en ML.
- ↗️ **Impacto:** Adquirir una ventaja competitiva clave en una industria donde el ML es el nuevo estándar.



El Ecosistema de TensorFlow.js: Herramientas y Recursos

TensorFlow.js está respaldado por un ecosistema robusto y bien documentado, proporcionando todos los recursos necesarios para aprender, experimentar y desplegar. Desde la documentación oficial y la referencia de la API hasta TensorFlow Hub, un vasto repositorio de modelos pre-entrenados, los desarrolladores tienen un arsenal de herramientas para acelerar drásticamente el ciclo de desarrollo.

- **Documentación Oficial:** Guías, demos y tutoriales.
<https://www.tensorflow.org/?hl=es-419>
- **Referencia de la API:** Consulta esencial de funciones y clases.
<https://js.tensorflow.org/api/latest/>
- **TensorFlow Hub:** Repositorio de modelos pre-entrenados por expertos.
<https://www.tensorflow.org/hub?hl=es>



El Poder de JS para ML: Flexibilidad y Ventajas Estratégicas

JavaScript es el único lenguaje con la flexibilidad para ejecutarse de forma nativa en prácticamente cualquier plataforma, desde el navegador hasta servidores, móviles e IoT.



Ventajas Estratégicas del ML en el Cliente



Privacidad

Los datos del usuario nunca abandonan su dispositivo, un punto crítico para la confianza y el cumplimiento de normativas.



Menor Latencia

El acceso directo a sensores (cámara, micrófono) y la ausencia de viajes al servidor permiten una interactividad en tiempo real.



Menor Costo

Se eliminan los costos de servidores con GPUs para inferencia. Solo se necesita hospedar los archivos estáticos.



Mayor Alcance y Escalabilidad

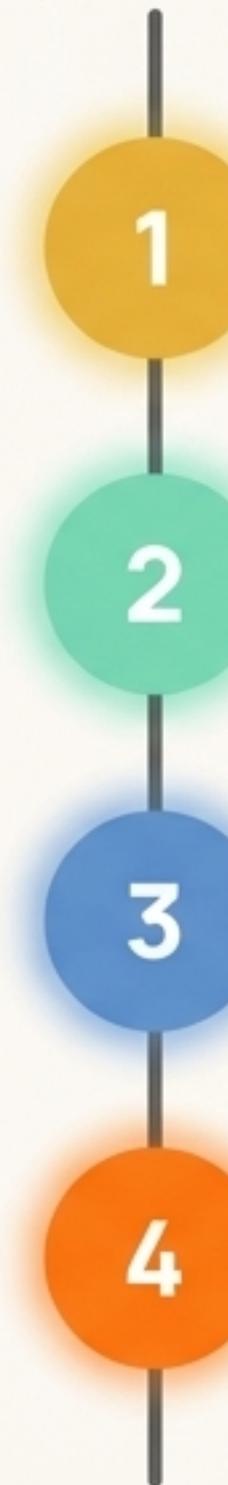
"Zero Install". Cualquier persona con un navegador puede acceder a la aplicación, alcanzando a miles de millones de usuarios.

Mapa de Aprendizaje: Fundamentos y Primeros Pasos (Capítulos 1-4)

Concepto Clave: La ruta de aprendizaje está estructurada para llevar a un desarrollador desde los conceptos fundamentales del ML hasta la creación de su primer modelo, entendiendo los bloques de construcción como los Tensores y las Neuronas.

Capítulo 2: Introducción al ML y TensorFlow.js

- Relación IA, ML y DL.
- Tipos de aprendizaje.
- Las 3 formas de usar ML.



Capítulo 1: Introducción al Curso

- Explorar el ML a través de la lente de JavaScript.
- Casos de uso reales.

Capítulo 3: Uso de Modelos Pre-hechos

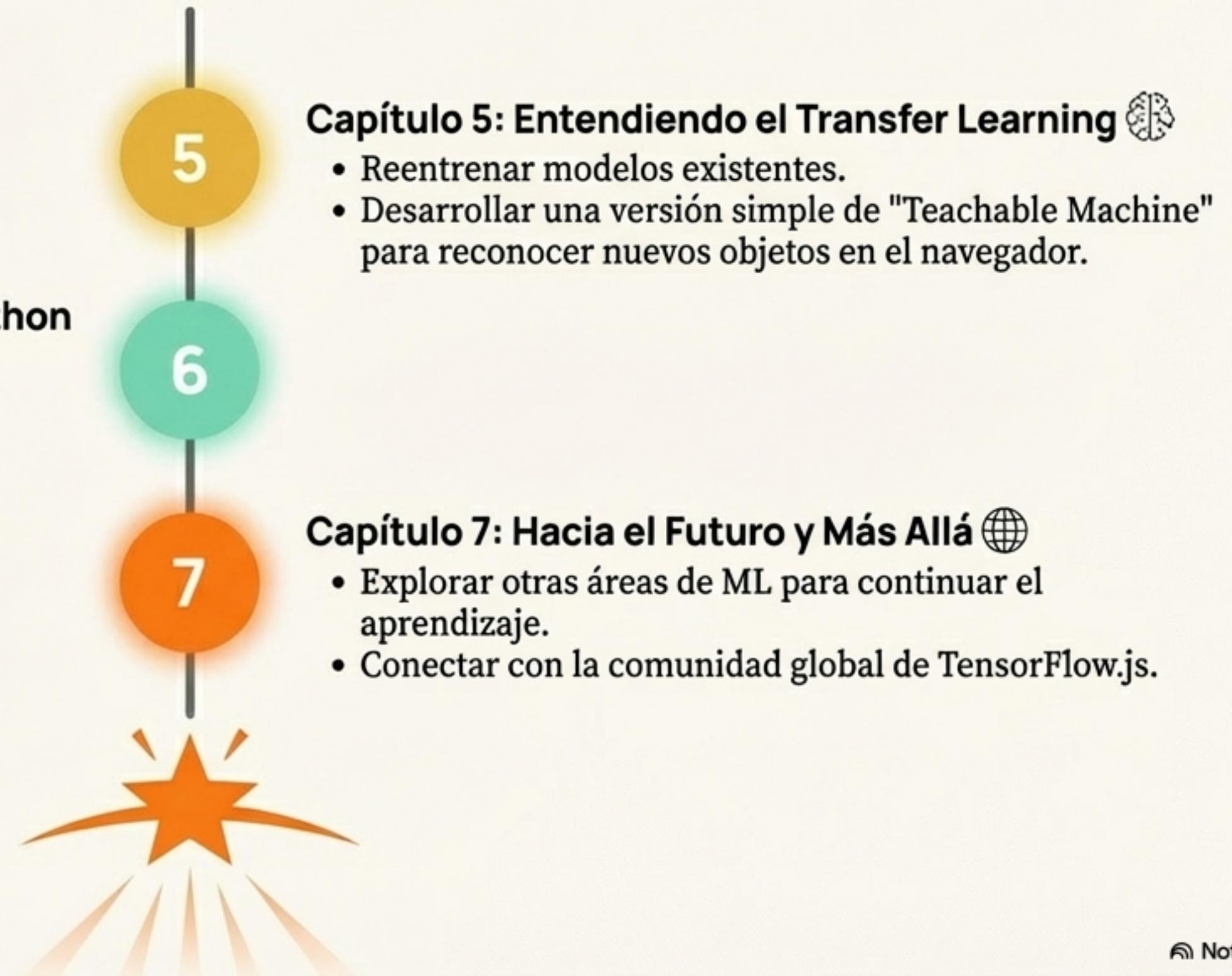
- Proyecto: cámara de seguridad inteligente.
- Introducción a los **Tensores**.
- Estimación de pose humana.

Capítulo 4: Escritura de Modelos Personalizados

- Recolección y división de datasets.
- Introducción a **Perceptrones/Neuronas**.
- Implementar **regresión lineal** y **clasificación**.

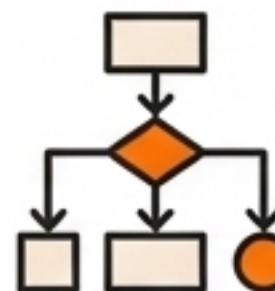
Mapa de Aprendizaje: Técnicas Avanzadas y Futuro (Capítulos 5-7)

Concepto Clave: Una vez dominados los fundamentos, el curso se adentra en técnicas avanzadas como Transfer Learning y la interoperabilidad con Python, concluyendo con una visión hacia el aprendizaje continuo y la comunidad global.



Un Nuevo Paradigma: Programación Tradicional vs. Machine Learning

Concepto Clave: La programación tradicional se basa en reglas explícitas codificadas por el desarrollador. El Machine Learning invierte este paradigma: se proporcionan datos y respuestas, y el algoritmo aprende las reglas por sí mismo, logrando una reusabilidad y adaptabilidad sin precedentes.



Programación Tradicional

Se escribe un programa con reglas explícitas a seguir. Dependiente de la lógica manual del programador.

```
if email contains 'V!agrà'  
    then mark as-spam;  
if email contains ...  
if email contains ...
```



Programas de Machine Learning

Se escribe un programa para aprender de ejemplos. El sistema encuentra patrones estadísticamente significativos.

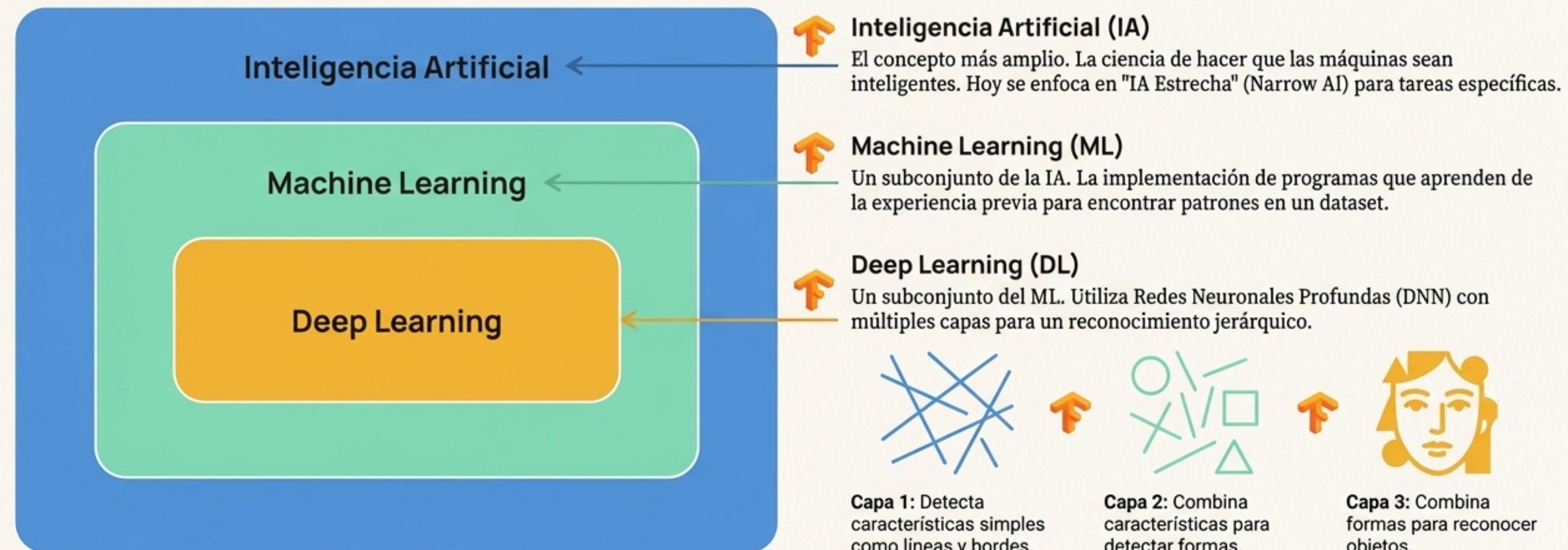
```
try to classify some emails;  
change self to reduce errors;  
repeat;
```

Ventaja Clave del ML: Reusabilidad y Adaptabilidad

Para pasar de un ‘reconocedor de gatos’ a un ‘reconocedor de perros’, no se cambia el código, se cambia el conjunto de datos de entrenamiento.

Taxonomía de la Inteligencia Artificial: IA, ML y Deep Learning

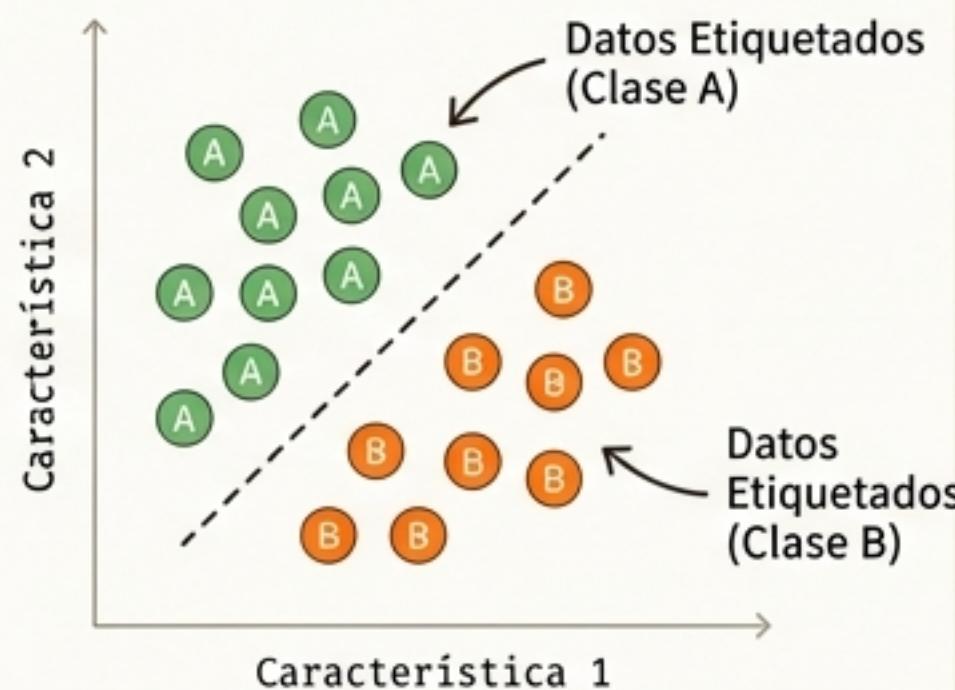
Concepto Clave: Estos términos representan una jerarquía conceptual clara. La IA es el campo más amplio, el ML es un subconjunto de la IA que aprende de datos, y el DL es una técnica de ML que utiliza Redes Neuronales Profundas para reconocer patrones complejos.



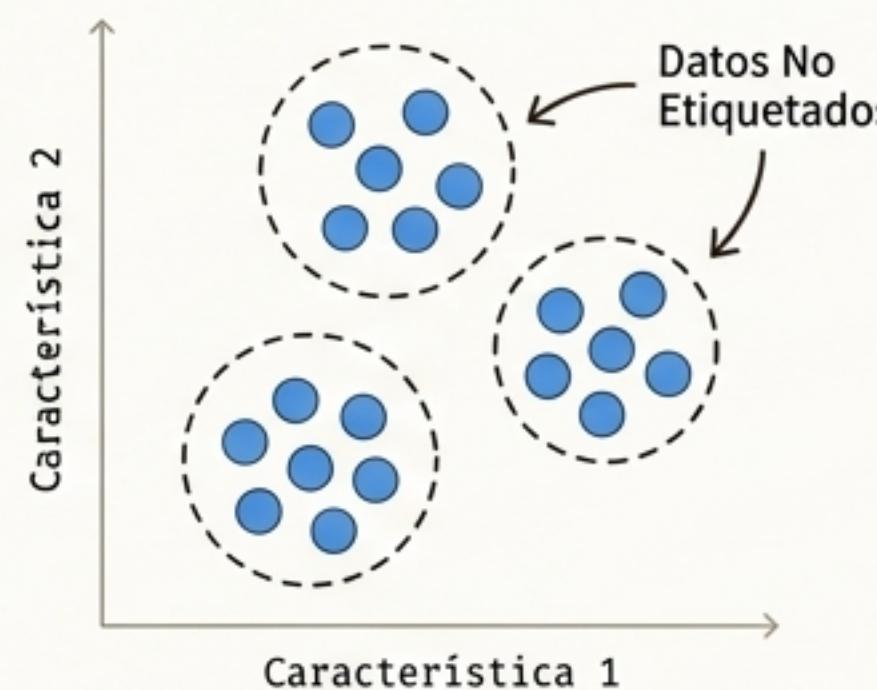
¿Cómo Aprenden las Máquinas?: Tipos de Aprendizaje

Concepto Clave: Existen tres paradigmas principales a través de los cuales los modelos de ML son entrenados, definidos por la naturaleza de los datos de entrada y el objetivo a alcanzar.

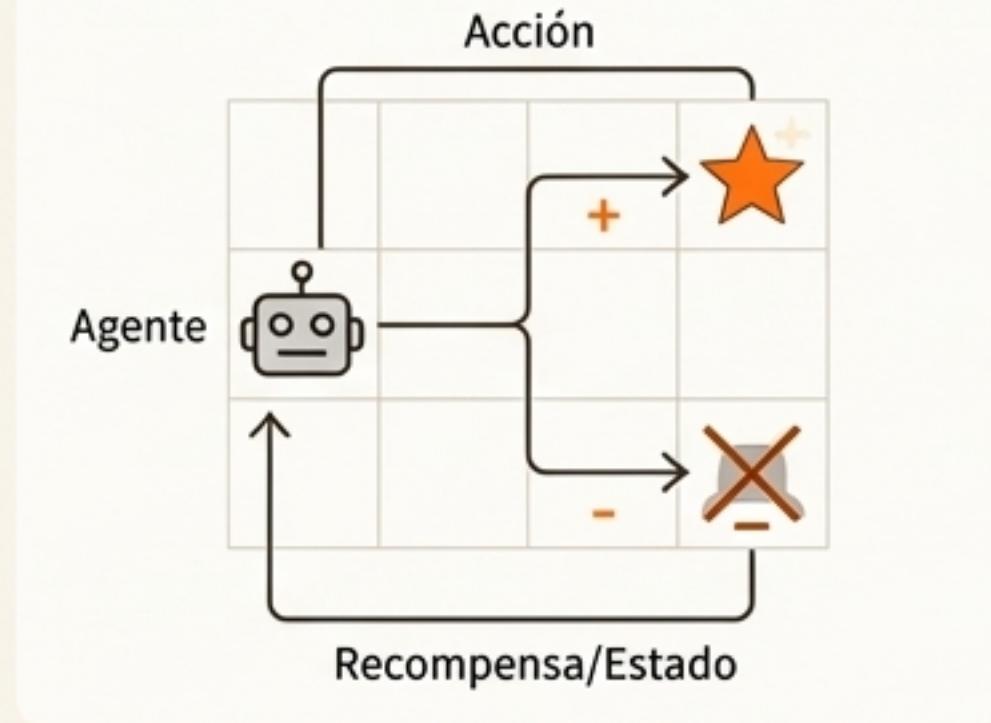
Supervisado



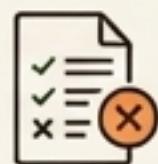
No Supervisado



Por Refuerzo



- **Datos:** Etiquetados (cada ejemplo tiene la 'respuesta correcta').
- **Objetivo:** Generalizar para predecir la etiqueta de datos nuevos.
- **Ejemplo:** Clasificar correos como spam/no-spam.



- **Datos:** No etiquetados.
- **Objetivo:** Descubrir patrones o clústeres inherentes en los datos.
- **Ejemplo:** Motores de recomendación que agrupan clientes.



- **Datos:** No hay dataset; el modelo aprende por interacción (prueba y error).
- **Objetivo:** Maximizar una 'recompensa' a lo largo del tiempo.
- **Ejemplo:** Entrenar una IA para jugar a un videojuego.

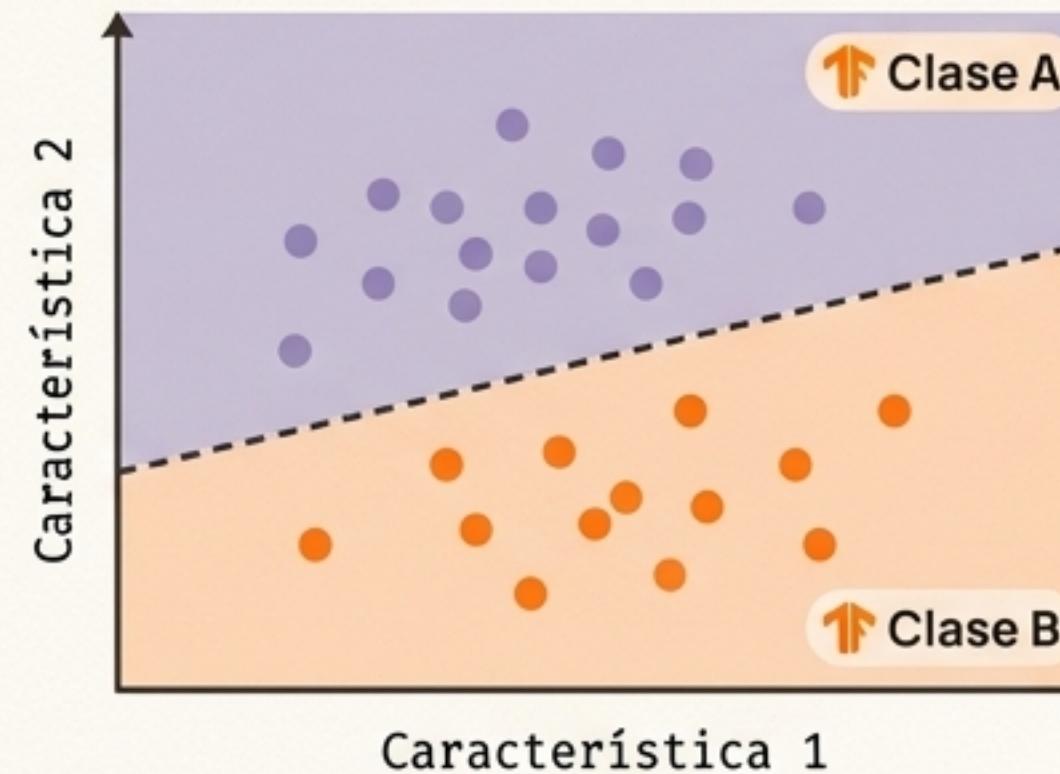


Resolviendo Problemas: Clasificación vs. Regresión

Concepto Clave: Dentro del aprendizaje supervisado, la Clasificación predice una categoría discreta (ej. “perro” o “gato”) encontrando un límite de decisión, mientras que la Regresión predice un valor numérico continuo (ej. “precio”) aprendiendo una función de mejor ajuste.

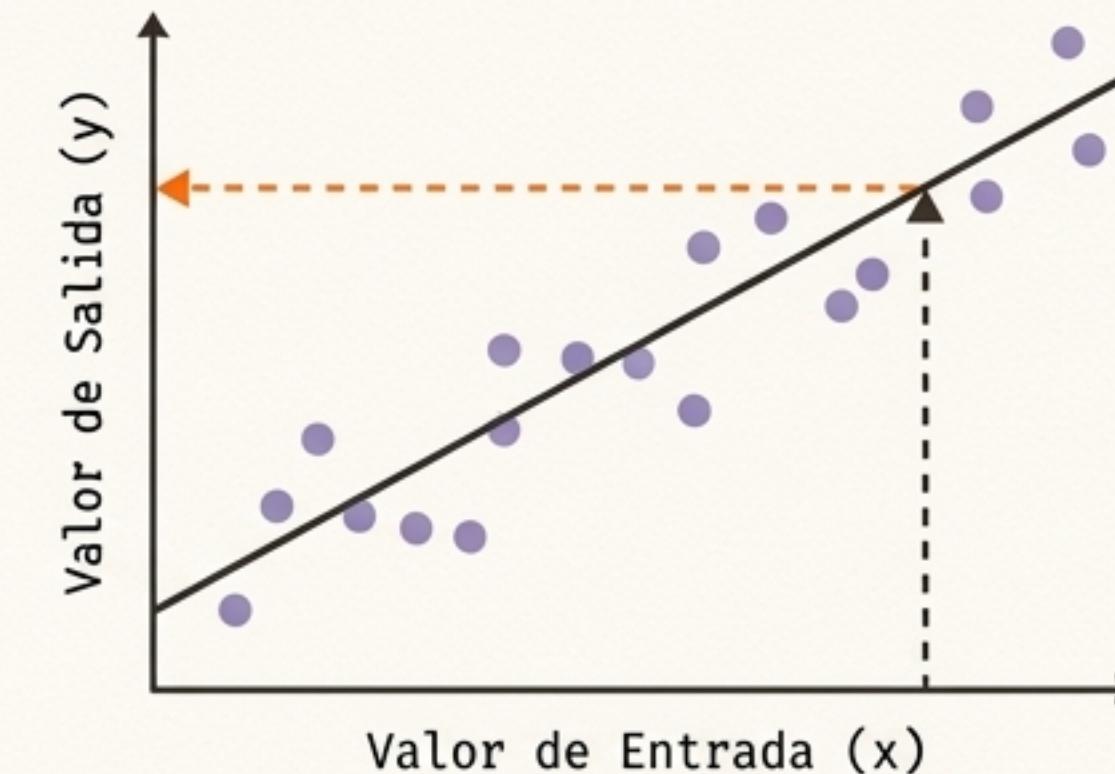
⬆️ Clasificación

- **Objetivo:** Predecir una clase o categoría.
- **Salida:** Una etiqueta discreta (ej. “manzana”, “naranja”, “spam”).



⬆️ Regresión

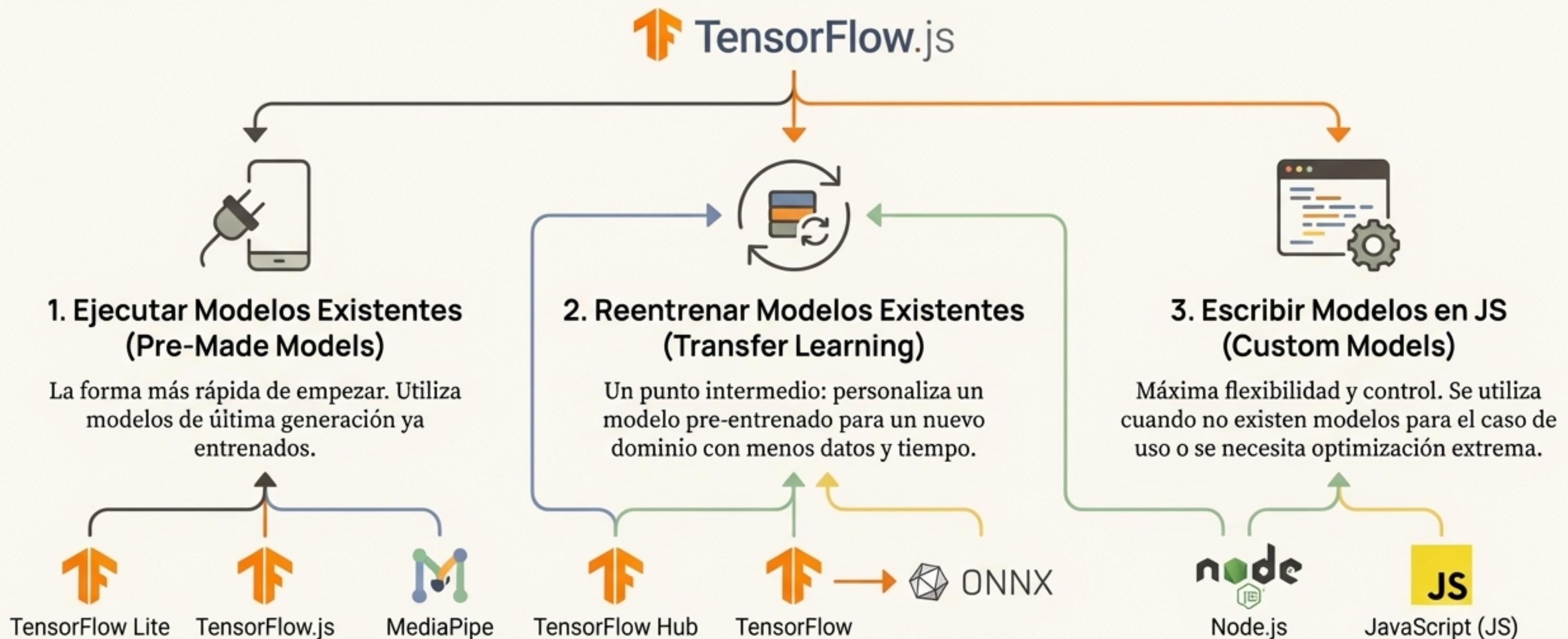
- **Objetivo:** Predecir un valor numérico continuo.
- **Salida:** Un número (ej. 250,000 €, 32.5°C).



Importancia de las Características (Features): En ambos casos, el éxito del modelo depende de seleccionar las características de entrada correctas (ej. color y peso).

Tres Caminos hacia la Implementación en TFJS

Concepto Clave: TensorFlow.js ofrece un espectro de opciones que se adaptan a diferentes niveles de experiencia, desde el uso de modelos pre-entrenados para una integración rápida hasta la escritura de modelos personalizados para un control total.



Aplicación Práctica: El Arsenal de Modelos Pre-entrenados

 **Concepto Clave:** TensorFlow.js proporciona una colección en expansión de modelos de última generación **listos para producción**, que abarcan dominios como Visión, Cuerpo Humano, PNL y Sonido, ahorrando tiempo y costos de entrenamiento.

tensorflow.org/js/models

Visión



- Clasificación de Imágenes
- Detección de Objetos

Cuerpo Humano



- Segmentación Corporal
- Estimación de Pose
- Detección de Hitos Faciales
- Estimación de Pose de Manos

Texto (NLP)



- Detección de Toxicidad
- Codificación de Sentencias
- BERT Q&A
- Detección de Intención

Sonido



- Reconocimiento de Comandos de Voz



Otros

- Clasificador KNN (K-Nearest Neighbors)

Fundamentos de TFJS: Los Tensores, el ADN de los Datos

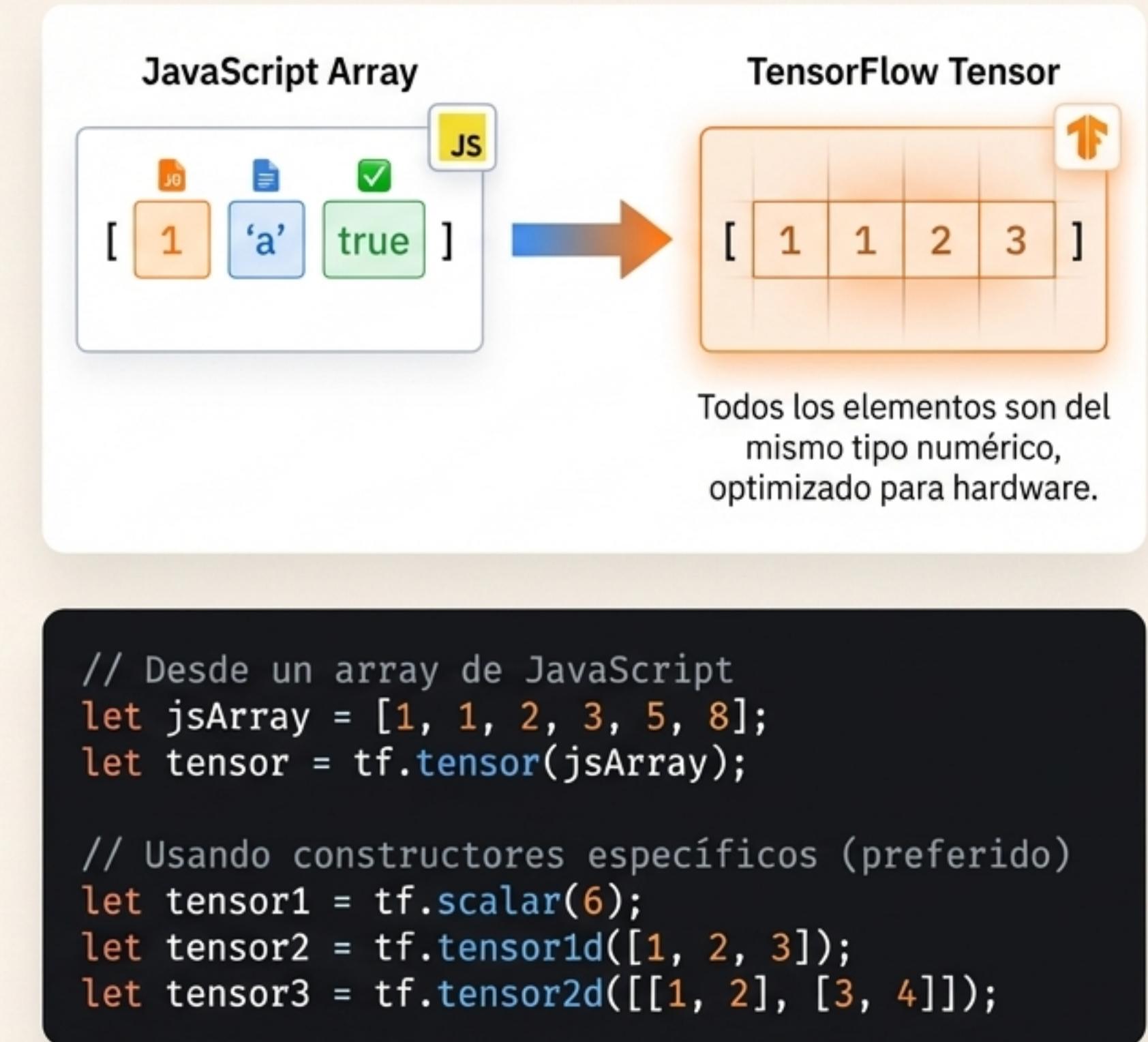
Concepto Clave

Los Tensores son la estructura de datos fundamental en TensorFlow. Son contenedores n-dimensionales para datos numéricos, respaldados por hardware (GPU) para operaciones masivamente paralelas. El nombre “TensorFlow” deriva del “flujo” de estos “tensores” a través del modelo.

- **Definición:** Similares a arrays, pero n-dimensionales y con un solo tipo de dato.
- **Inmutabilidad:** Las operaciones crean nuevos tensores; no modifican los existentes.
- **Flujo de Datos:** Los modelos toman tensores como entrada, realizan operaciones y producen tensores como salida.

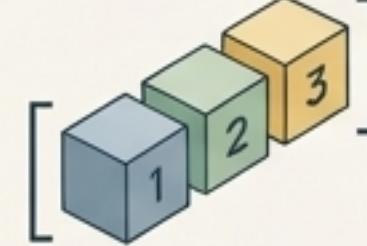
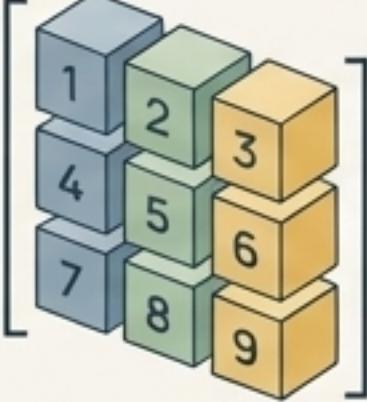
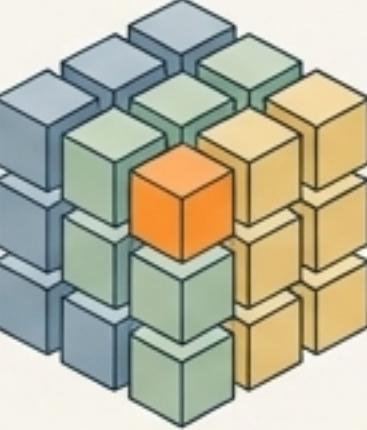
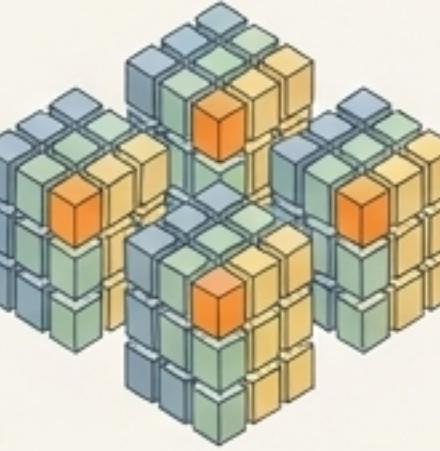
Terminología Clave:

- **Rango (Rank):** Número de dimensiones.
- **Forma (Shape):** Tamaño de cada dimensión.
- **Tipo de Dato (DType):** El tipo numérico (`float32`, `int32`).



Anatomía de un Tensor: Rango, Forma y Tipos de Datos

Cada tensor se define por tres propiedades: **Rango** (número de dimensiones), **Forma** (tamaño de cada dimensión), y **Tipo de Dato** (ej. `float32`), que son críticos para la memoria y el rendimiento.

Rango 0: Escalar	Rango 1: Vector	Rango 2: Matriz	Rango 3: Cubo de Datos	Rango 4: Lote de Imágenes	Tipo de Dato (DType)
					Define el tipo numérico de los datos: `float32` (precisión), `int32` (enteros), `bool` (booleano). Afecta directamente el uso de memoria.
Rango: 0 Forma: [] Uso: Un solo valor	Rango: 1 Forma: [3] Uso: Coordenadas	Rango: 2 Forma: [3, 3] Uso: Imagen en escala de grises	Rango: 3 Forma: [3, 3, 3] Uso: Imagen a color (RGB)	Rango: 4 Forma: [N, 3, 3, 3] Uso: Un lote (batch) de imágenes	

Operaciones y Rendimiento con Tensores

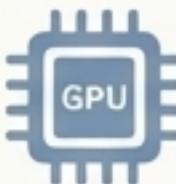
Los tensores incluyen operaciones matemáticas (`.mul()`, `.add()`). Se usan en lugar de JS nativo para delegar los cálculos a backends de hardware optimizados (GPU/CPU), logrando una velocidad órdenes de magnitud superior.

```
// Tensor inicial: shape [2, 3]
let tensor = tf.tensor2d([1, 2, 3], [4, 5, 6]);

// Multiplicación en paralelo usando la GPU
let scalar = tf.scalar(2);
let newTensor = tensor.mul(scalar);
// Resultado: [[2, 4, 6], [8, 10, 12]]

// Cambiar la forma del tensor (sin costo computacional)
let reshaped = newTensor.reshape([6]);
// Resultado: [2, 4, 6, 8, 10, 12]
```

Backends de Hardware para Ejecución Paralela



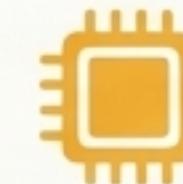
WebGL (GPU)

Ejecución rápida en GPU. Ideal para modelos grandes (>10MB). Soportado por el 97.6% de dispositivos.



WASM (CPU)

Ejecución rápida en CPU (SIMD & multithreading). Ideal para modelos pequeños (<10MB).



CPU (Fallback)

El backend más lento, pero siempre disponible como fallback.



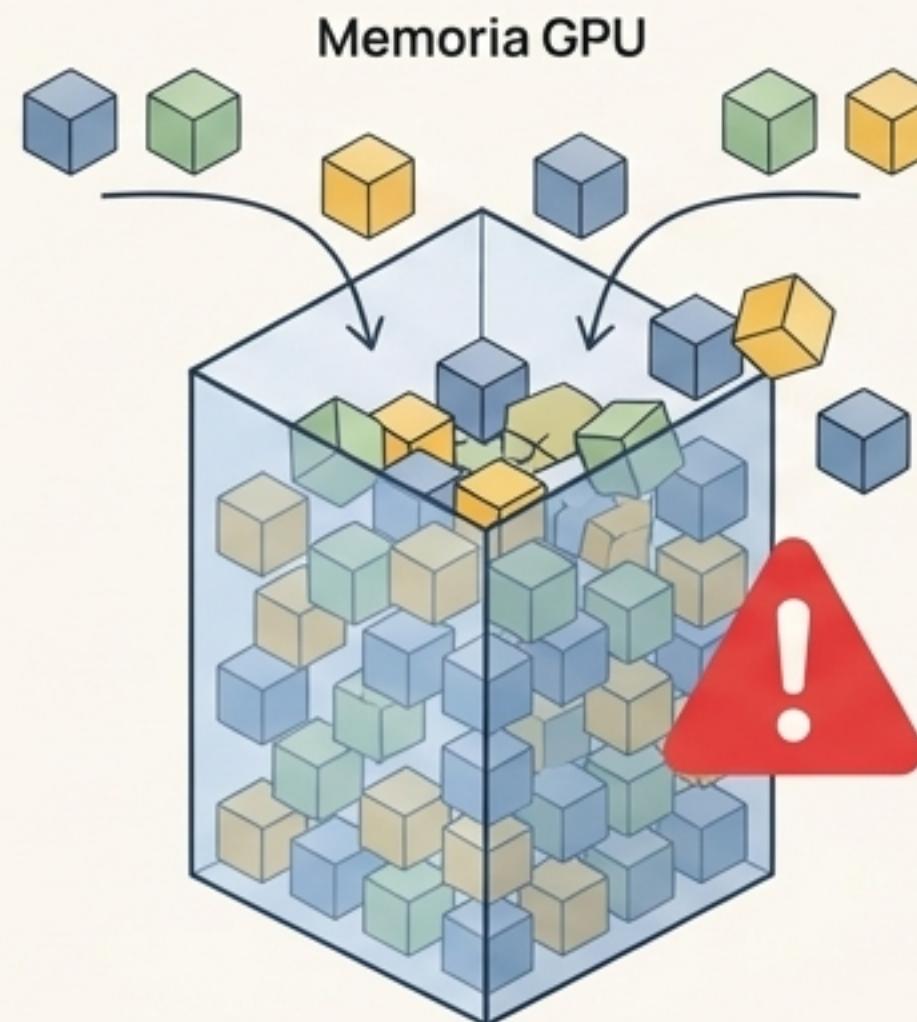
Futuro (WebGPU / WebNN)

Nuevos estándares web para una ejecución nativa aún más rápida.

El Problema - Fugas de Memoria

Gestión de Memoria en TFJS: La Responsabilidad del Desarrollador

A diferencia de los objetos JS, la memoria de los Tensores (especialmente en WebGL) no se libera automáticamente. El desarrollador debe "disponer" (dispose) de ellos para evitar fugas de memoria (*memory leaks*) que pueden bloquear la aplicación.



La Solución - Gestión Activa

Solución: `tf.tidy()`

`tf.tidy()` es una utilidad que limpia automáticamente todos los tensores intermedios creados dentro de su alcance. Es el método recomendado para evitar fugas.

```
// tf.tidy() limpia todos los tensores excepto el que se retorna.
function miOperacion(inputTensor) {
  return tf.tidy(() => {
    // Tensores intermedios creados aquí
    const temp1 = inputTensor.mul(tf.scalar(2));
    const temp2 = temp1.add(tf.scalar(5));

    // Solo 'temp2' sobrevive fuera del tidy.
    // 'temp1' será dispuesto automáticamente.
    return temp2;
  });
}
```

