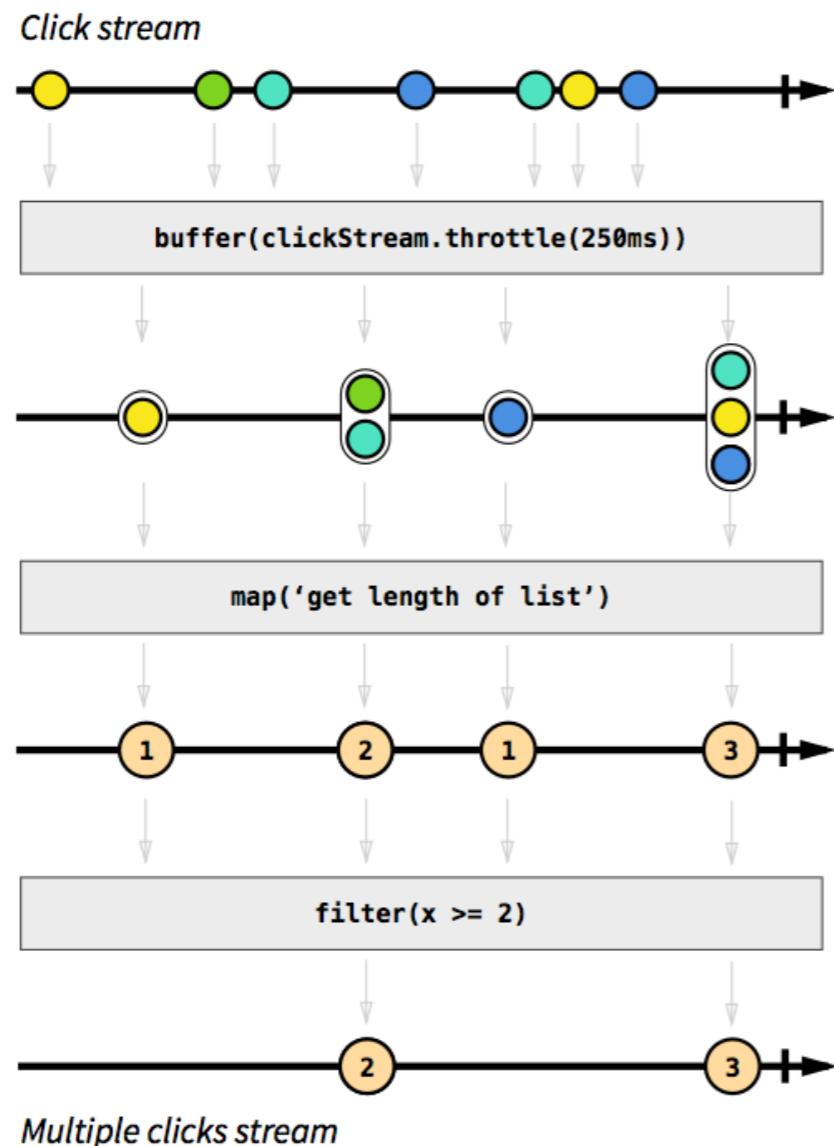


# Tema 7: El “futuro” del Desarrollo Web Frontend

# Functional Reactive Programming

- Un paradigma en el que los eventos se modelan como **streams** a los que se puede aplicar un **pipeline** de **operadores**



<https://rxjs.dev/>



<https://baconjs.github.io/>



<https://mobx.js.org>

# Algunas ideas ¿con futuro?

(y también presente)

- **Webassembly:** ejecutar cualquier lenguaje en el navegador



- **HTMX:** HTML “a lo grande” en cuanto a la H



high power tools for HTML

- **IA en el navegador**

```
const writer = await ai.writer.create();
const result = await writer.write("Write a email asking for feedback");
```

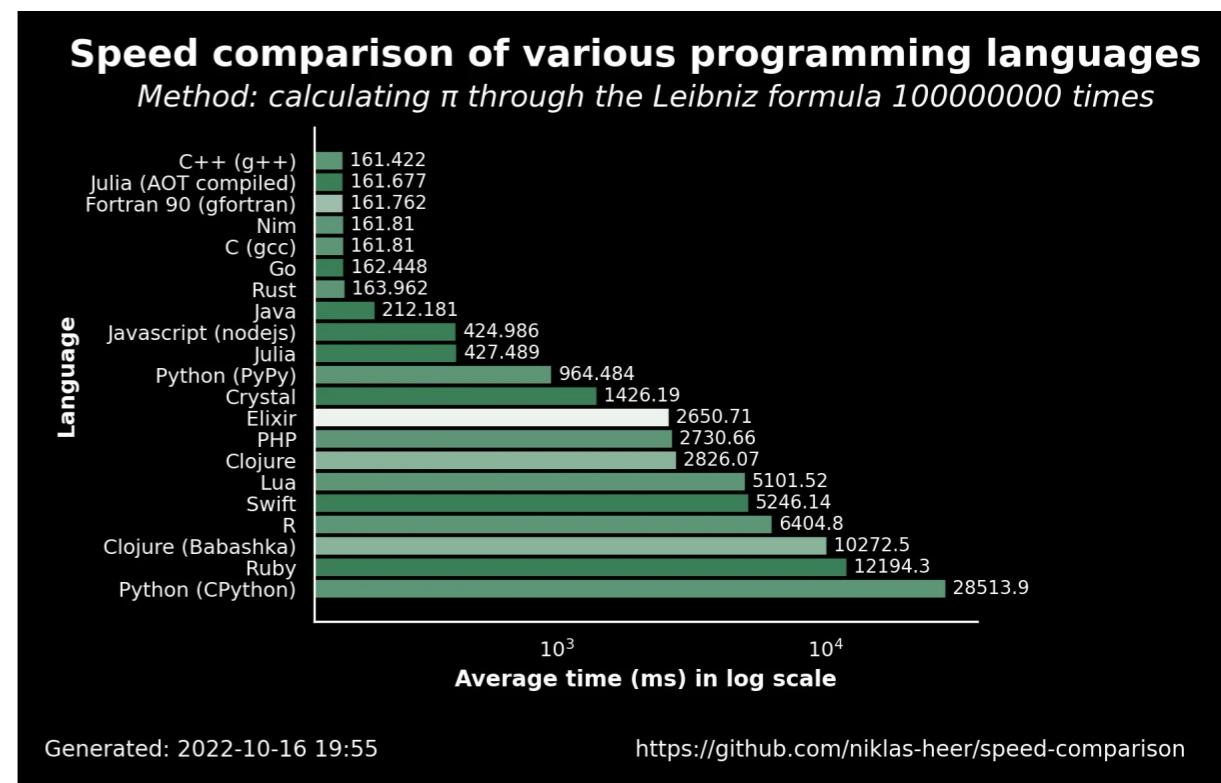
Tema 7 : El “futuro” del Desarrollo Web Frontend

7.1

WebAssembly

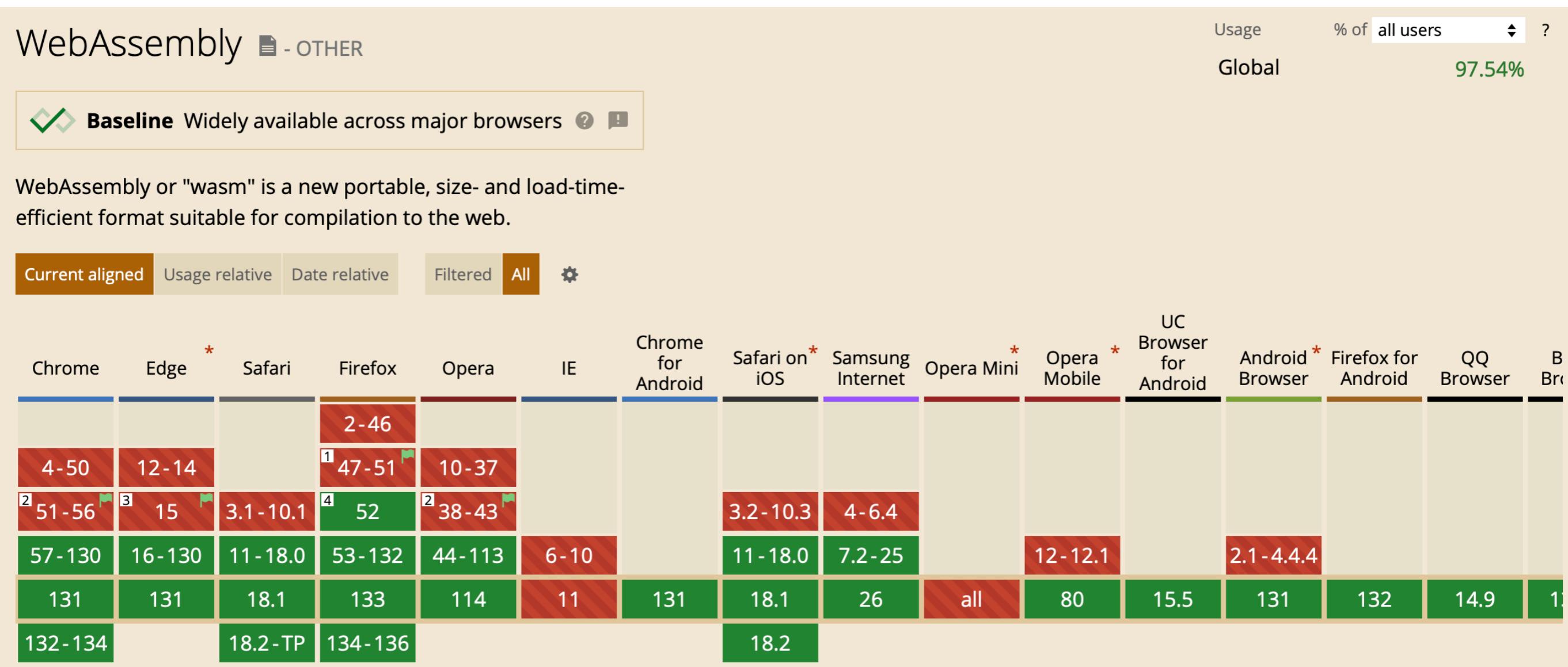
# ¿Por qué WebAssembly?

- **Javascript** es un lenguaje bastante potente pero no es precisamente el más rápido
- **WebAssembly (WASM)**: un formato de código binario para una máquina virtual, diseñado como objetivo de compilación de lenguajes de alto nivel como C++, Go, Rust, ....
- **Objetivos:**
  - Rendimiento cercano al nativo
  - Portabilidad de código
  - Seguridad (sandboxing en el navegador)



```
1 (module
2   (type $type1 (func (param i32) (result i32)))
3   (type $type2 (func (param i32))) ; inner types in module
4   (import "env" "sqrt"           ; Import the sqrt function
5     (func $sqrt (type $type1))) ; function signature is type1
6   (import "env" "print"         ; Import the print function
7     (func $print (type $type1))) ; function signature is type2
8   (func $main                  ; define a function
9     (type $type1)              ; function signature is type1
10    (local $var i32)           ; define a local variable
11    loop                      ; Enter a separate stack space
12      block                   ; Enter a separate stack space
13        local.get $var          ; Push the first param on stack
14        call $sqrt              ; Call imported sqrt function
15        local.set $var          ; Pop return value to $var
16    end
17    local.get $var              ; Push return value on the stack
18    br_if 1                   ; Out of the execution of the function
19    local.get $var              ; The param of the $print function
20    i32.const 1                ; Push the element id of the function to be called
21    call_indirect (type $type2) ; Signature of the function is type2
22    br 0                      ; Jump to line 11. Restart external loop
23  end)
24  (table 5 5 funcref)
25  (elem (i32.const 1) func $print)) ; The first element points $print
```

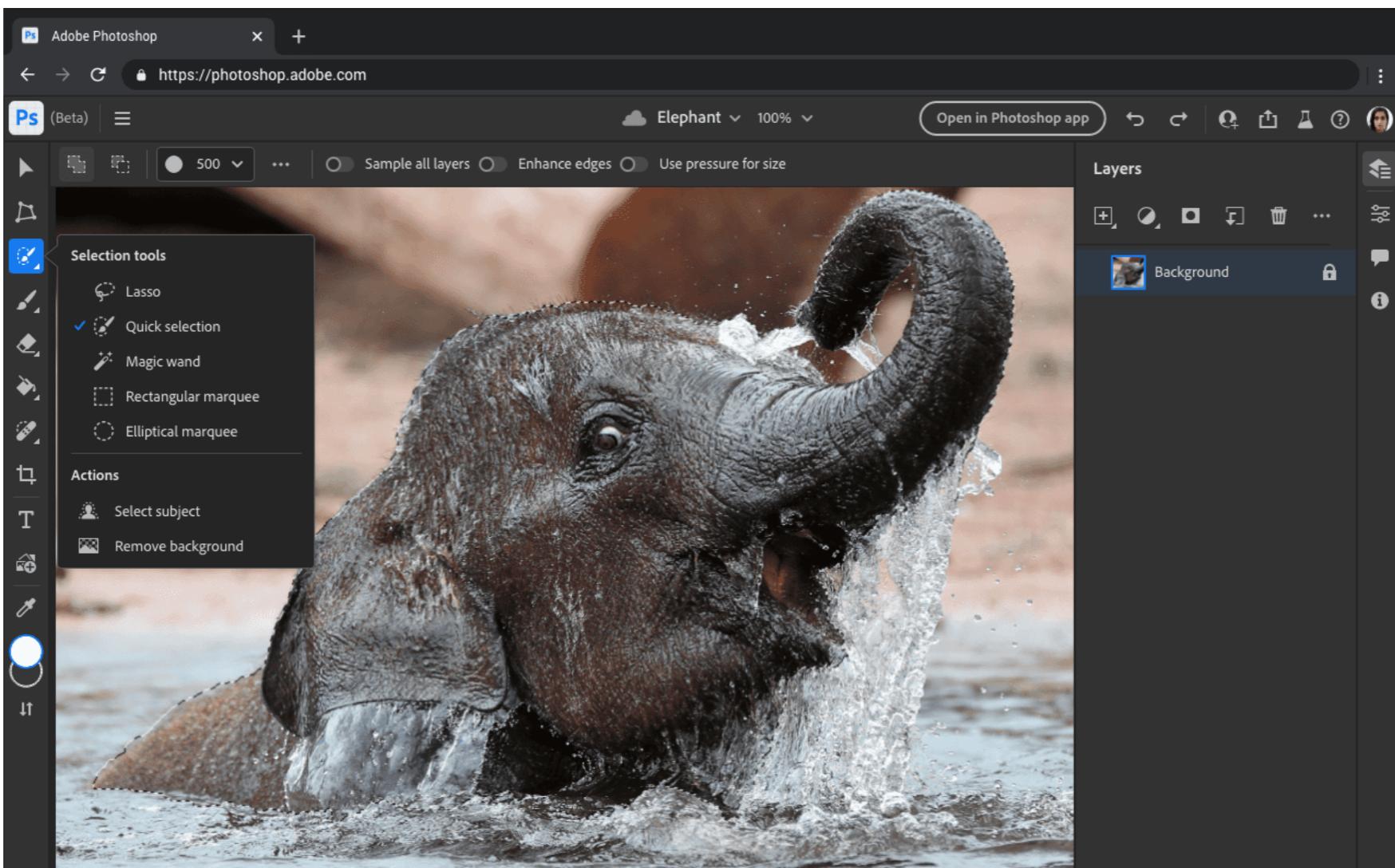
# WebAssembly ya es el presente



<https://caniuse.com/wasm>

# Photoshop en la web

- Photoshop se pudo portar al navegador gracias a **WebAssembly**
- Usa otras funcionalidades como **web components** (componentes UI) o **service workers** (para cache de recursos)
- **Más info:**
  - [Artículo de Addy Osmani en Medium](#)
  - [Artículo de web.dev](#)



# Muchos otros ejemplos

- **Pyodide**: Python en el navegador
- **Google Earth**
- **Unity** permite exportar juegos a la web con webassembly
- **AutoCAD**
- **Figma**
- ...

<https://madewithwebassembly.com/>

# Ejemplo de herramientas: Emscripten



# emscripten

**Emscripten is a complete compiler toolchain to WebAssembly, using LLVM, with a special focus on speed, size, and the Web platform.**

## Porting

Compile your existing projects written in C or C++ – or any language that uses [LLVM](#) – to browsers, [Node.js](#), or [wasm runtimes](#).

## APIs

Emscripten converts OpenGL into WebGL, and has support for familiar APIs like SDL, pthreads, and POSIX, as well as Web APIs and JavaScript.

## Fast

Thanks to the combination of LLVM, Emscripten, [Binaryen](#), and [WebAssembly](#), the output is compact and runs at near-native speed.

<https://emscripten.org/>

# Demo Fibonacci en WASM vs JS

```
#include <emscripten/bind.h>
```

```
int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n-1) + fibonacci(n-2);
}
```

```
EMSCRIPTEN_BINDINGS(module) {
    emscripten::function("fibonacci", &fibonacci);
}
```

fibonacci.c



```
em++ fibonacci.c -o fibonacciWASM.js -s WASM=1 -s
NO_EXIT_RUNTIME=1 -s
"EXPORTED_RUNTIME_METHODS=['ccall']" -s
EXPORT_NAME="Module" -s MODULARIZE=1 --bind
```



fibonacciWASM.js  
fibonacciWASM.wasm

```
let wasmModule;
```

```
Module()
  .then(function (module) {
    wasmModule = module;
    console.log('WebAssembly module loaded');
  })
```

```
const wasmStart = performance.now();
const wasmResult = wasmModule.fibonacci(n);
const wasmTime = performance.now() - wasmStart;
```

index.js

<https://stackblitz.com/edit/vitejs-vite-clqcdr?file=index.js>

# FFmpeg en WebAssembly

- **FFmpeg** es una herramienta en línea de comandos para procesar audio y video (transcodificar, cortar, aplicar filtros,...), desarrollada en C++
- **ffmpeg.wasm** es la conversión a WASM

## ffmpeg.wasm

ffmpeg.wasm is a pure WebAssembly / JavaScript port of FFmpeg enabling video & audio record, convert and stream right inside browsers!

Try it Now!

<https://ffmpgewasm.netlify.app/docs/getting-started/usage>

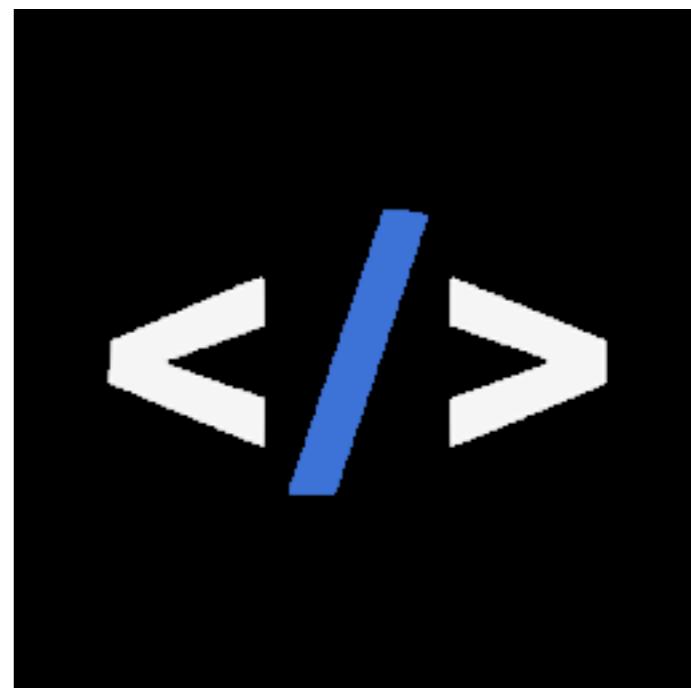
Tema 7 : El “futuro” del Desarrollo Web Frontend

## 7.2 HTMX

O cómo podría ser que el futuro sea volver al pasado

# ¿Qué es HTMX?

- Una **librería JS** en el *frontend* para ampliar las capacidades hipermedia de HTML
- En HTML convencional, solo **unas pocas etiquetas** son **hipermedia**, y además están limitadas
  - <a> (solo peticiones GET, la respuesta sustituye toda la página actual)
  - <form> (solo peticiones GET y POST, la respuesta sustituye a toda la página actual)



<https://htmx.org/>

¿Y si pudiéramos añadir **funcionalidades hipermedia** a **cualquier etiqueta**? ¿Y que además fueran **menos limitadas**? (por ejemplo, al clicar en un botón lanzar una petición HTTP de tipo POST y que la respuesta se coloque en una etiqueta HTML ya existente)

"

```
<button hx-post="/like" hx-trigger="click"
         hx-target="#output">
    ¡Me gusta!
</button>
<div id="output"></div>
```

# Las 4 generalizaciones de HTMX

- Hipermedia para:
  - Cualquier **etiqueta** (no solo <a> y <form>)
  - Cualquier tipo de **petición** (no solo GET o POST)
  - Cualquier **evento** (hay algunos que ya son por defecto, como click en botones)
  - Que la respuesta **sustituya cualquier elemento** de HTML
- Algunos atributos tienen modificadores

```
<input type="text" name="q"
      hx-get="/trigger_delay"
      hx-trigger="keyup changed delay:500ms"
      hx-target="#search-results"
      placeholder="Search...>
<div id="search-results"></div>
```

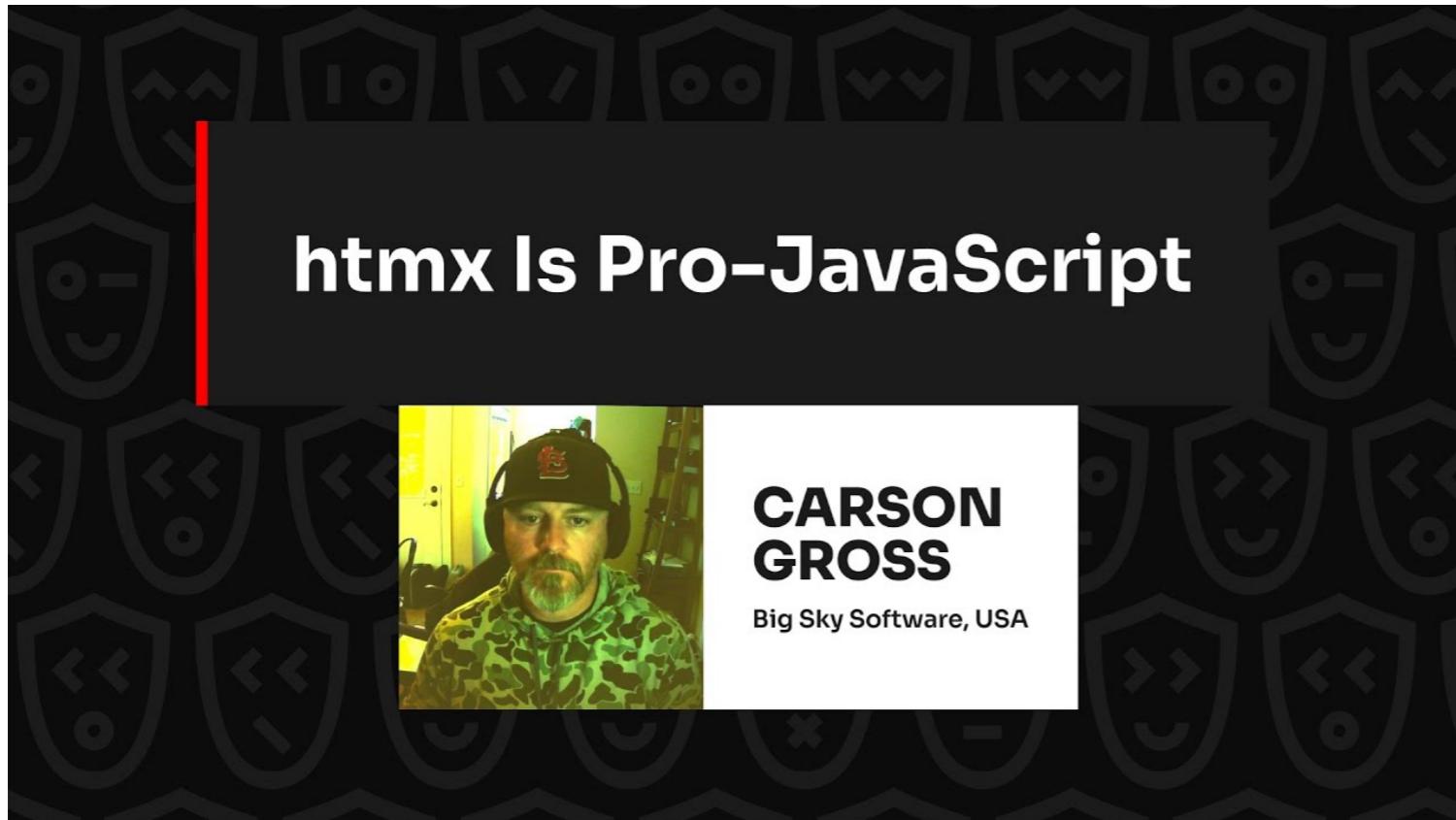
<https://htmx.org/examples/active-search/>

# Filosofía de uso

- Objetivo: poder crear interfaces de usuario al estilo SPA **reduciendo al mínimo el JS**
- JS pasa de ser el centro de la app a usarse solo en determinadas funcionalidades, para **extender las capacidades de la web** (como se concibió el lenguaje inicialmente)

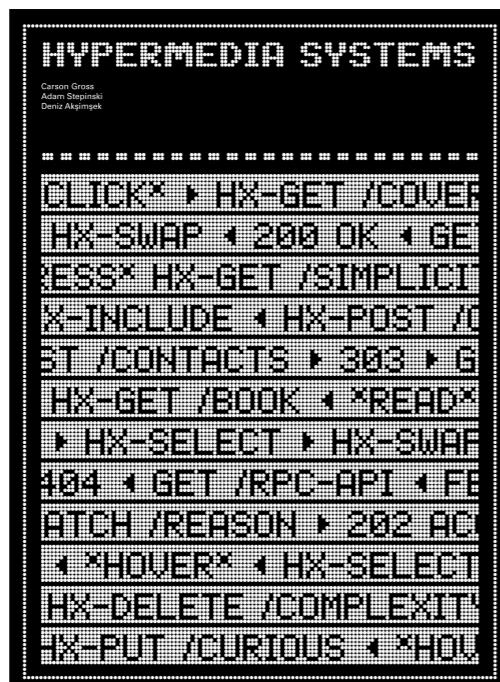
<https://htmx.org/examples/sortable/>

# Para saber más...



Presentación en la conferencia JSNation 2024 de Carson Gross, el creador de HTMX, en la que habla de HTMX y su relación con JS

<https://youtu.be/9ZhmnfKD5PE>



Un libro escrito por el propio Carson Gross en el que habla con detalle de HTMX y la filosofía de las apps desarrolladas con él. Se puede leer online de manera gratuita pero también se vende en formato físico

<https://hypermedia.systems/>

## Tema 7 : El “futuro” del Desarrollo Web Frontend

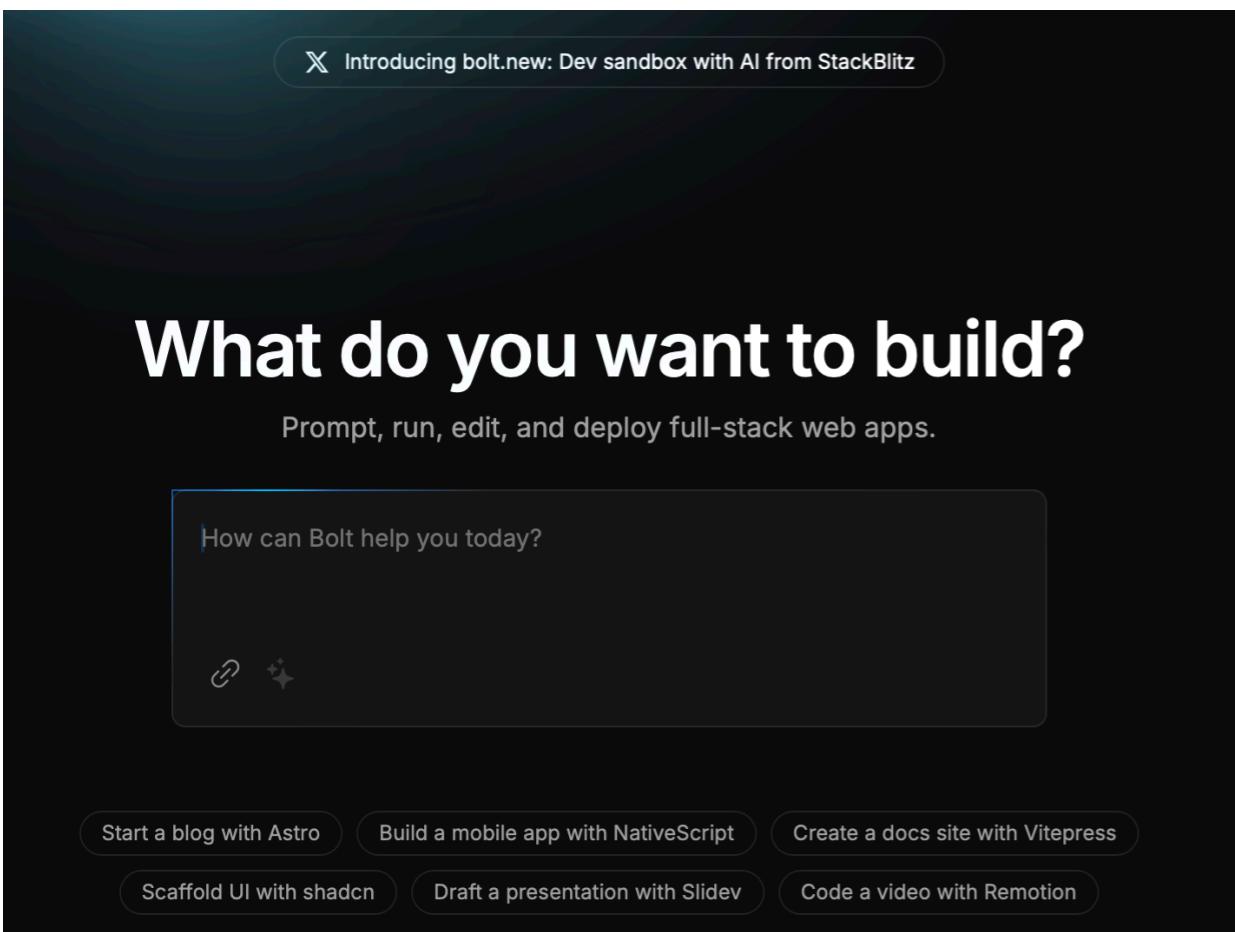
### 7.3

## IA en el navegador

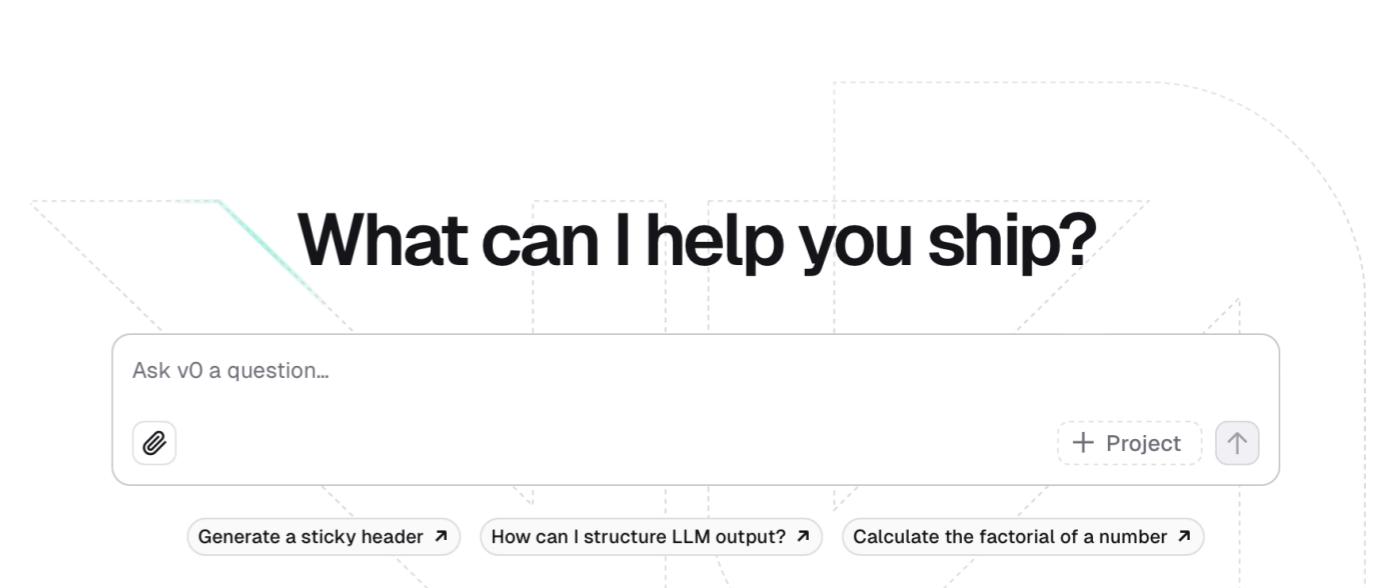
O cómo una vez más se cumple la “Ley de Atwood”: “Cualquier aplicación que se pueda escribir en JavaScript, se acabará escribiendo en JavaScript”

# Desarrollo web con IA

**Bolt** (<https://bolt.new/>)



**v0** (<https://v0.dev/>)



# IA en la nube vs. en el navegador



- Ventajas de la IA en el navegador
  - Privacidad
  - Funcionamiento sin conexión
  - Mejor experiencia de usuario (sin latencia)
  - Acceso a la aceleración por hardware (si la hay : )
  - Cero instalación

# Soporte actual en el navegador

- **Soporte nativo** (todavía en fase de desarrollo):
  - **Chrome** ofrece herramientas de IA integrada para trabajar con textos (traducción, resumen, escritura de textos...). Usa en local el LLM Gemini Nano (versiones de 1.8B o 3.2B parámetros, cuantizado a 4 bits)
  - **Microsoft Edge** ofrece WebNN, un API para ejecutar redes neuronales en el navegador
- **Bibliotecas de terceros:** hay unas cuantas, por ejemplo
  - Transformers.js
  - Tensorflow.js
  - Mediapipe web

# Transformers.js

- Como su propio nombre indica, una librería para ejecutar redes neuronales basadas en la arquitectura **transformer**
  - Los **transformers** se han aplicado en **múltiples campos**: LLMs, visión (clasificación de imágenes, detección de objetos, segmentación,...), audio (reconocimiento de voz, generación de habla,...),...
- Orientada a **inferencia**, no a entrenamiento
- Funcionalmente equivalente a la librería python [transformers](#) (salvo entrenamiento)

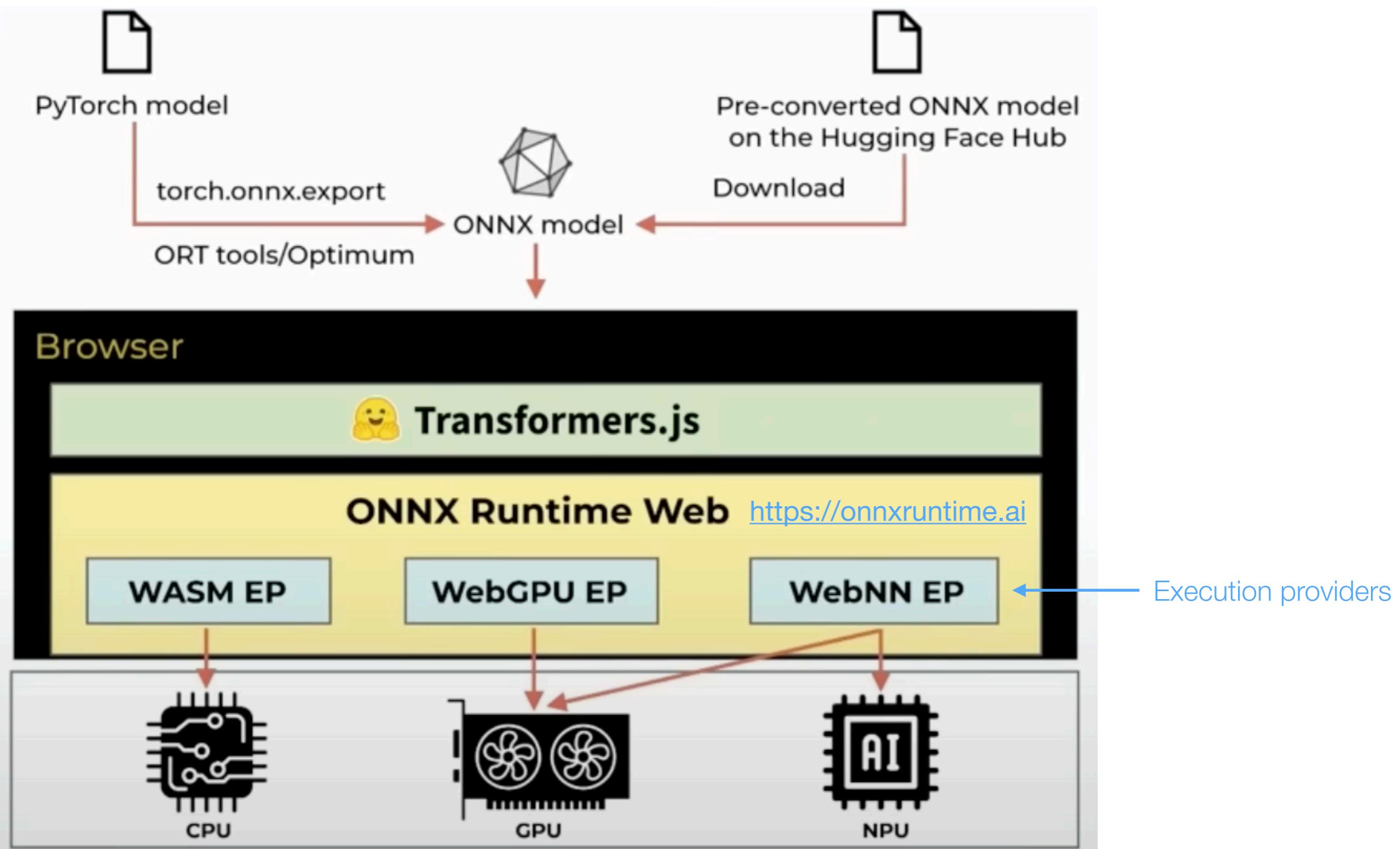


<https://huggingface.co/docs/transformers.js/index>

# Qué se puede hacer con transformers.js

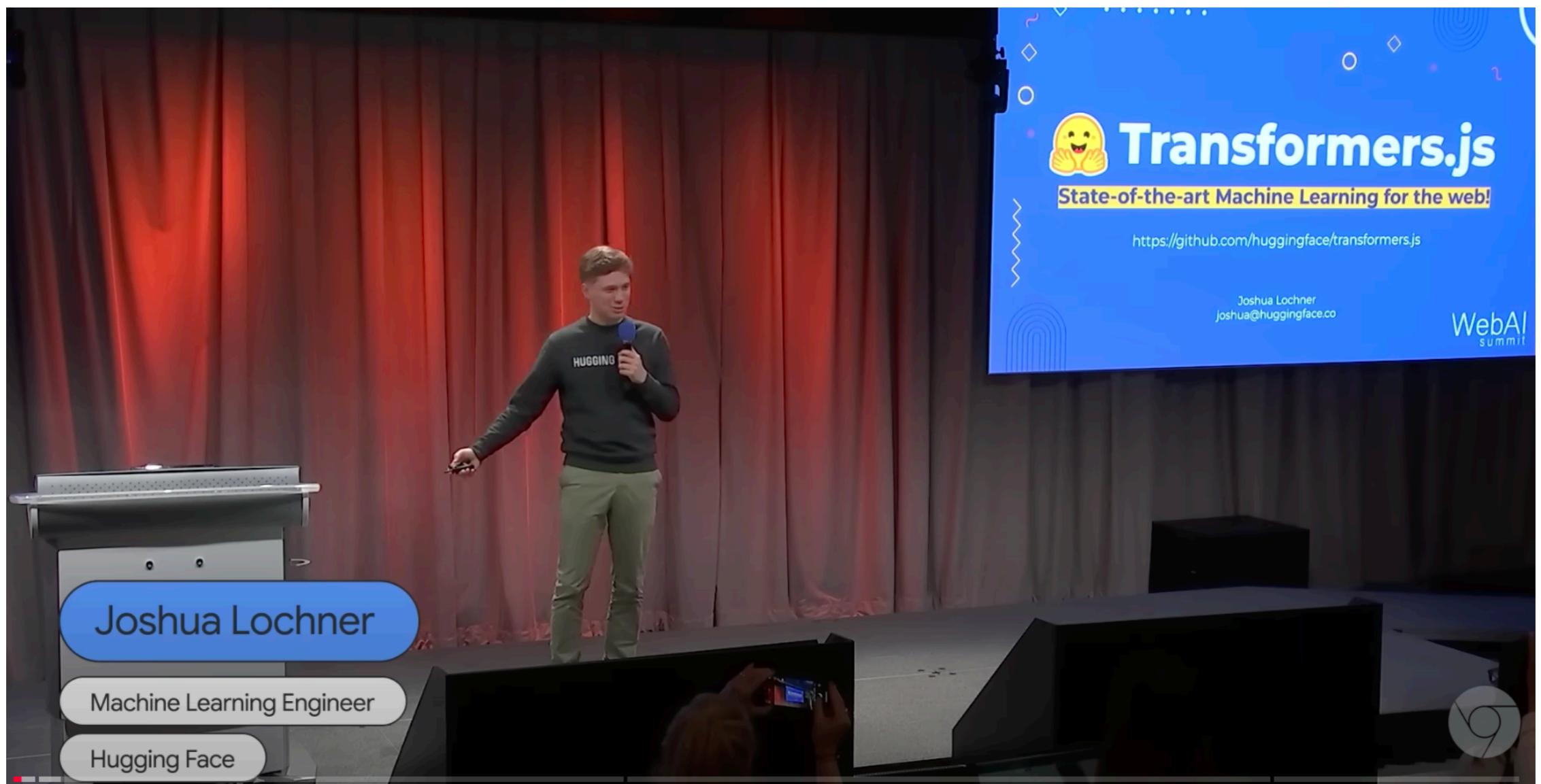
- Colección de Demos
  - Modelos del lenguaje en local acelerados con WebGPU
  - Modelos de lenguaje multimodales (imagen, texto)
  - Clasificación de textos y de imágenes
  - Análisis de sentimientos
  - Reconocimiento de objetos (un juego de doodle basado en ello)
  - Reconocimiento y síntesis de voz
  - ...

# Transformers.js por dentro



# Para saber más...

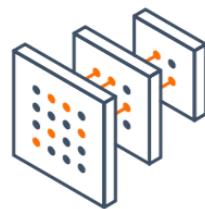
Transformers.js: State-of-the-art Machine Learning for the web: una charla del creador de la librería en la Google Web AI Summit 2024, con una explicación clara de la misma y muchas demos con enlaces



# Tensorflow.js

- <https://www.tensorflow.org/js> : un *port* de la biblioteca python del mismo nombre, permite realizar inferencias y también entrenar modelos
- A diferencia de transformers se usa para cualquier tipo de red neuronal

## Cómo funciona



### Ejecutar modelos existentes

Utilice modelos de JavaScript disponibles en el mercado o convierta modelos de Python TensorFlow para ejecutarlos en el navegador o en Node.js.



### Vuelva a entrenar los modelos existentes

Vuelva a entrenar modelos de ML preexistentes utilizando sus propios datos.



### Desarrollar ML con JavaScript

Cree y entrene modelos directamente en JavaScript utilizando API flexibles e intuitivas.

Utilice modelos oficiales de  
TensorFlow.js →

Convertir modelos de Python →

Utilice Transfer Learning para personalizar  
modelos →

Comience con TensorFlow.js →

# Algunas aplicaciones de tensorflow.js

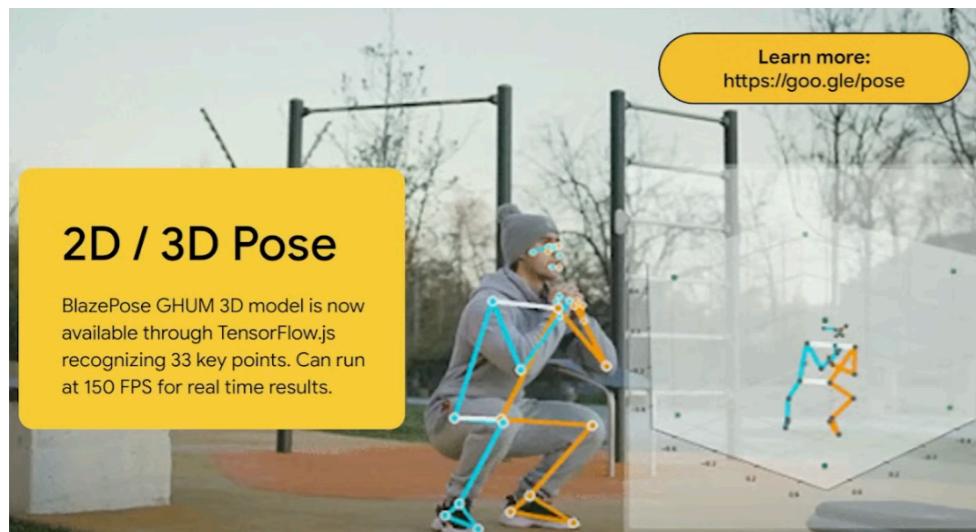
## Población

Vea ejemplos y demostraciones en vivo creadas con TensorFlow.js.

The image displays three cards from the TensorFlow.js website's population section. Each card has a title, a brief description, and two buttons at the bottom: 'Explorar demostración' and 'Ver código'.

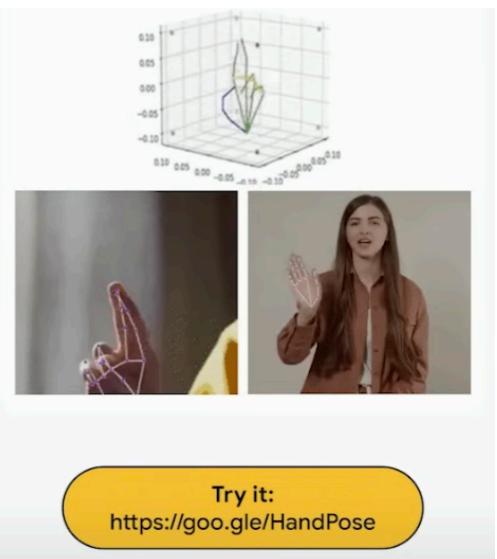
- holocabina**: Transpórtate a una playa tropical, al espacio exterior y a otros lugares con el poder del aprendizaje automático web.
- Sincronización de labios de YouTube**: Mira qué tan bien te sincronizas con la letra del popular éxito "Dance Monkey". Esta experiencia en el navegador utiliza el modelo Facemesh para estimar puntos clave alrededor de los labios y obtener precisión en la sincronización de labios.
- Búsqueda del tesoro de emojis**: Usa la cámara de tu teléfono para identificar emojis en el mundo real. ¿Podrás encontrar todos los emojis antes de que acabe el tiempo?

<https://www.tensorflow.org/js/demos?hl=es>



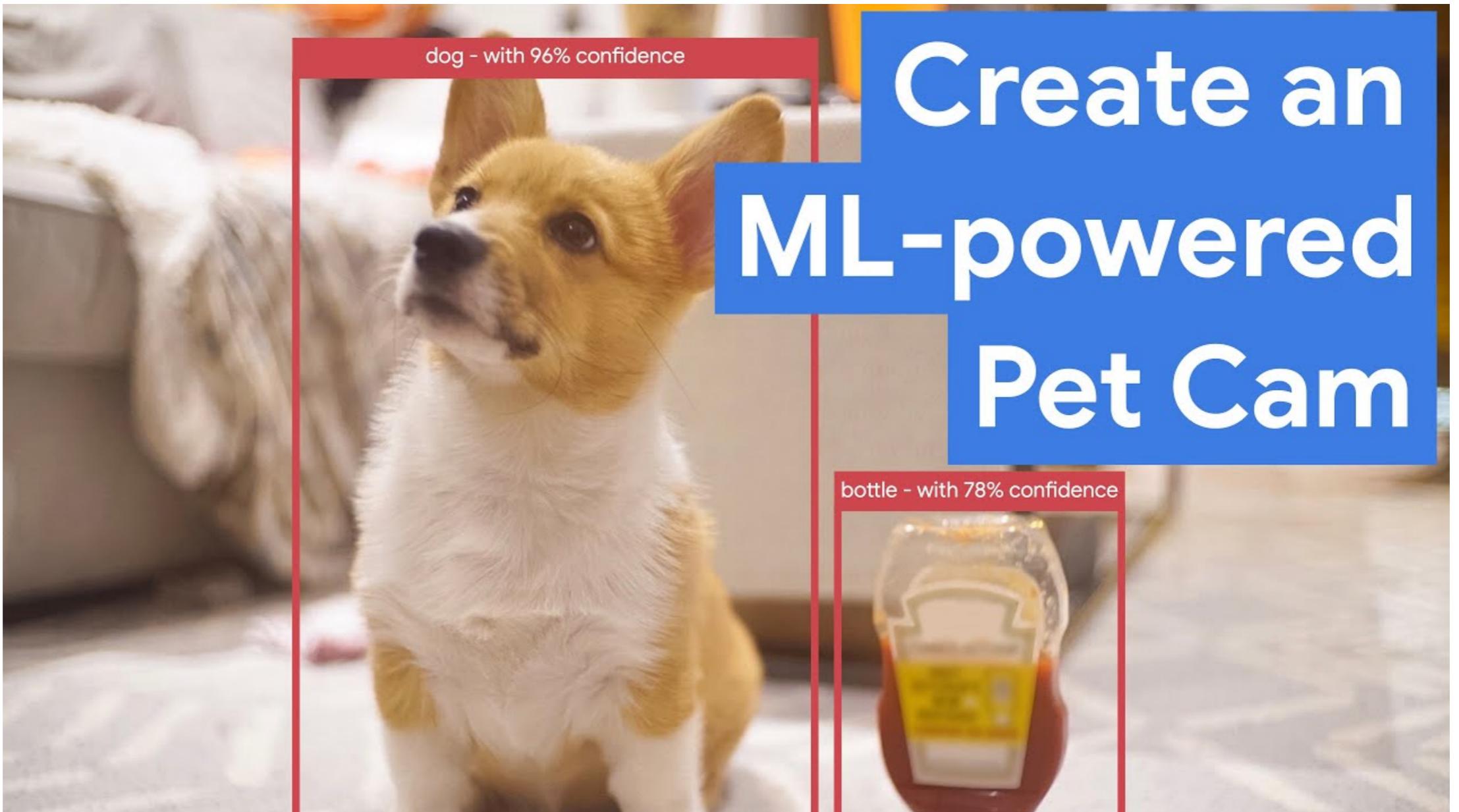
## Hand Pose Estimation

Track multiple hands in 3D or 2D and with higher precision than before.



<https://goo.gle/pose>

<https://goo.gle/HandPose>



<https://goo.gle/petcam>

Tutorial con código de ejemplo de cómo desarrollar un sistema para detectar si una mascota se sale de su espacio asignado o se acerca demasiado a un objeto (p.ej. el sofá 😊)

# Mediapipe web

- Mediapipe es un conjunto de bibliotecas y herramientas para desarrollar apps con visión artificial (esqueletos, reconocimiento objetos, segmentación), procesamiento de audio (reconocimiento de voz, ...), texto (LLMs,...)
- Mediapipe web es la versión JS para ejecutar en navegador
- Usa Tensorflow “por debajo”

## Algunas demos

<https://pjbelo.github.io/mediapipe-js-demos/>

<https://ai.google.dev/edge/mediapipe/solutions/examples?hl=es-419>

# IA integrada en Chrome

- De momento orientada solo al procesamiento de texto con un LLM local (Gemini Nano)

```
const {  
    available,  
    defaultTemperature,  
    defaultTopK,  
    maxTopK  
} = await ai.assistant.capabilities();  
  
if (available == "yes") {  
    // Crear una nueva sesión con el asistente de AI  
    const session = await ai.languageModel.create();  
    // Pedir al asistente que escriba un poema  
    const result = await session.prompt("Escribe un poema sobre las aplicaciones web");  
    // Mostrar el resultado en la consola  
    console.log(result);  
}
```

Del [blog de Lino Uruñuela](#), aunque en el artículo usa el API “antiguo” (de sept 2024) `ai.assistant` en lugar de `ai.languageModel`