

BUILD A ZERO-COST AI AGENT IN YOUR BROWSER



Ship a Live Agent Today

WHO AM I



 /in/nishantkthakur

Nishant Thakur

Adobe · Bengaluru · 15 years in tech

 github.com/nkthakur48



Full-Stack + AI



MS Data Science



Curious builder



Believes in eternal learning

\$0

WHAT IF AI COST YOU NOTHING?



No API keys



No servers



No cloud bills

Today you build one.

THE STACK



**Gemma 2
2B**

Google's open
model · 1.5 GB



WebLLM
OpenAI-
compatible API



WebGPU
GPU compute
in browser



**Your
HTML**
No npm · No
build step

HOW IT WORKS

QUANTIZATION

Weights
compressed to
4-bit - multi-
GB model →
~1.5 GB

WEBGPU

Successor to
WebGL for
general
compute -
GPU inference
in the browser

WEBLLM

OpenAI-
compatible API
- one CDN
import, no
install

That's all you need to know. Let's build. →



PHASE 1

The AI Loop

From blank file to streaming chat in 40 minutes

QUICK CHECK

-  Chrome open · `chrome://gpu` → **WebGPU:**
Hardware accelerated
-  VS Code + Live Server ready
-  Model cached from prework (should load instantly)

 Model not cached? Start `precache.html` now -
you'll catch up by the streaming step.

STEP 1 - SCAFFOLD + LOAD THE MODEL

```
<!DOCTYPE html>
<html>
<body>
  <p id="status">Loading model...</p>
  <script type="module">
    import { CreateMLCEngine } from
      "https://esm.run/@mlc-ai/web-llm";

    const statusEl = document.getElementById("status");

    const engine = await CreateMLCEngine(
      "gemma-2-2b-it-q4f16_1-MLC", {
        initProgressCallback: ({ text }) => {
          statusEl.textContent = text;
        }
      }
    );
    statusEl.textContent = "Model ready!";
  </script>
</body>
</html>
```

One import. One function call. That's the whole loader.

WHAT YOU JUST WROTE



All of this is running **inside your browser tab**

STEP 2 - FIRST CHAT COMPLETION

```
const response = await engine.chat.completions.create({
  messages: [
    { role: "user", content: "What is WebGPU?" }
  ],
});

document.getElementById("output").textContent =
  response.choices[0].message.content;
```

Run it. Change the prompt. Run it again.

That's it - AI in your browser.

STEP 3 - STREAMING

```
const chunks = await engine.chat.completions.create({
  messages: chatHistory,
  stream: true,           // ← one flag ▶
});

let reply = "";
for await (const chunk of chunks) {
  reply += chunk.choices[0]?.delta.content || "";
  outputEl.textContent = reply;  // tokens appear live
}
```

💡 Same API as OpenAI. If you know one, you know both.

STEP 4 - PERSONAS

```
const chatHistory = [  
  {  
    role: "system",  
    content: "You are a pirate. Keep responses to 2-3 sentences."  
  }  
];
```

Swap the persona live:



Pirate



Shakespeare



Scientist



5-year-old

Your AI, your rules.

ADD MEMORY (MULTI-TURN)

```
// Before sending:  
chatHistory.push({ role: "user", content: userText });  
  
// After receiving:  
chatHistory.push({ role: "assistant", content: reply });
```

Push to the array, send the full history. Follow-ups just work.

🚩 Behind? Grab [checkpoint.html](#) - it has everything so far.



PHASE 2

Make It Real

From raw text to a chat app in 20 minutes

BEFORE → AFTER

🔍 Checkpoint: Browser AI

Model ready!

You: What is GPU?

Gemma: GPU stands for **Graphics Processing Unit**. It's a specialized electronic circuit that handles the processing of graphics and image data. It's like a mini computer dedicated to visuals.

You: Tabulate clear differences between GPU and CPU

Gemma: ## GPU vs. CPU: Key Differences

Feature	GPU	CPU

Primary function Handles visual processing (graphics) Handles general-purpose computing		
Cores Hundreds to thousands, specialized Hundreds to thousands, general-purpose		
Architecture Highly parallel, for parallel tasks Sequential, for sequential tasks		
Power consumption Higher power consumption Lower power consumption		
Speed Slower clock speeds for single tasks Faster clock speeds for general tasks		
Memory Larger dedicated memory (VRAM) Smaller shared system memory (RAM)		

In short: GPUs are better at handling graphics and parallel tasks, while CPUs are better at general-purpose tasks.

Type a message...

Send

Phase 1 output

AI Assistant

LOCAL GPU Clear Chat

Hi! I'm your AI assistant running entirely in your browser. No servers, no API keys. How can I help?

What is GPU?

GPU stands for Graphics Processing Unit. It's a specialized electronic circuit designed to accelerate the creation of images and special effects in computer graphics. Essentially, it's the "brain" behind the visual content you see on your computer.

38 tok/s - on-device

Tabulate clear differences between GPU and CPU

GPU vs CPU: Key Differences

Feature	GPU	CPU
Primary Function	Graphics processing and visual effects	General-purpose processing
Cores	Multiple cores for parallel processing	Fewer cores for sequential processing
Architecture	Optimized for parallel tasks	Optimized for sequential tasks
Memory	Dedicated VRAM for fast data access	Shared system RAM for data access
Power Consumption	High power consumption for parallel tasks	Lower power consumption for sequential tasks
Examples	GeForce, Radeon GPUs	Intel Core, AMD Ryzen CPUs

In essence:

- CPU is like the "general worker" handling diverse tasks.
- GPU is like a "special worker" focused on graphics and visuals.

54 tok/s - on-device

Type a message...

Send

What you'll build now

CHAT LAYOUT - HTML + CSS

```
<div id="chat-messages">
  ←!— messages appear here →
</div>

<div class="input-bar">
  <input type="text" id="input"
    placeholder="Type a message...">
  <button id="send">Send</button>
</div>
```



Paste style.css

Bubbles, scroll container, input bar - all handled.

WIRE JS TO THE UI

```
function addMessage(role, text) {  
  const div = document.createElement("div");  
  div.className = `message ${role}`;  
  div.textContent = text;  
  chatMessages.appendChild(div);  
  chatMessages.scrollTop = chatMessages.scrollHeight;  
  return div;  
}
```

On send:

1. Append user bubble → show "Thinking..."
2. Stream tokens into assistant bubble → auto-scroll
3. Disable input during generation

LOADING + POLISH

```
const engine = await CreateMLCEngine(  
  "gemma-2-2b-it-q4f16_1-MLC", {  
    initProgressCallback: ({ text, progress }) => {  
      loadingStatus.textContent = text;  
      if (progress !== undefined) {  
        progressEl.style.width = (progress * 100) + "%";  
      }  
    }  
  }  
);  
loadingEl.style.display = "none";  
appEl.style.display = "flex";
```

Test it - you should have a styled, working chat app. 🎉



PHASE 3

Make It Yours

Customize · Then see what's next

YOUR AGENT, YOUR RULES

```
const AGENT_NAME    = "FitCoach";  
const SYSTEM_PROMPT = `You are a certified fitness coach.  
    Be encouraging. Ask about goals before prescribing.  
    Keep answers to 2-3 sentences.`;  
const WELCOME_MSG   = "Hey! Ready to crush your goals? 💪";
```

Pick yours:



Fitness coach



Travel guide



Language tutor



Chef



Budget advisor



Game master

Change title, header, bubble colors to match. Test 3–4 turns.
Refine.

PROMPT ENGINEERING - 3 RULES

1. BE SPECIFIC ABOUT TONE + LENGTH

"Keep responses to 2–3 sentences unless asked for detail"

2. GIVE EXAMPLES OF DESIRED OUTPUT

"For a recipe: ingredients list
→ numbered steps"

3. SET BOUNDARIES

"Never give medical diagnoses. Suggest consulting a professional."

BUT WAIT - WHAT IF YOUR AI COULD *DO* THINGS?

Search Wikipedia · Check the weather · Do math
· Call APIs

Live demo incoming

LIVE DEMO: RESEARCH AGENT

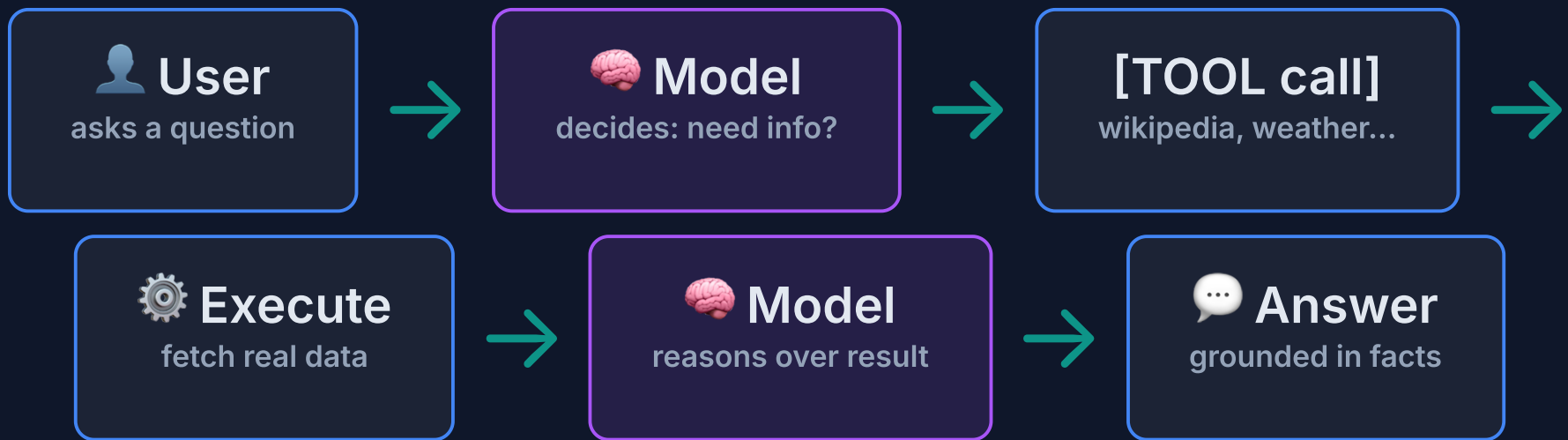
An agent that **calls tools** and reasons over results.

 Ask a factual question → watch it call Wikipedia
→ synthesize an answer

Try: "How tall is the Eiffel Tower?" or "What's the weather in Tokyo?"

→ open `research-agent.html`

THE AGENTIC LOOP



The model loops until it has enough information - or hits
MAX_TOOL_ROUNDS

THE CODE - TOOL-CALL PARSING

```
// Model outputs: [TOOL wikipedia "Eiffel Tower"]
function parseToolCall(text) {
  const match =
    text.match(/\[TOOL\s+(\w+)(?:\s+"([^"]*)"?)?\]/i);
  if (!match) return null;
  return { tool: match[1].toLowerCase(), arg: match[2] || "" };
}
```

```
// The loop
for (let round = 0; round < MAX_TOOL_ROUNDS; round++) {
  const { reply } = await getModelResponse();
  const toolCall = parseToolCall(reply);
  if (!toolCall) break; // no tool → final answer

  const result = await executeTool(toolCall.tool, toolCall.arg);
  chatHistory.push({
    role: "user",
    content: `[TOOL RESULT]\n${result.text}\n[/TOOL RESULT]`
  });
}
```

WHAT ELSE YOU CAN BUILD

Same pattern: structured markers in model output → parse → render

CHOOSE YOUR ADVENTURE

[CHOICE_1] markers
→ clickable buttons

QUIZ TUTOR

[CORRECT]/[WRONG]
→ live score tracking

BUDGET TRACKER

[ADD groceries 45] →
expense table

FLASHCARDS

[FRONT]...[BACK] →
3D flip cards

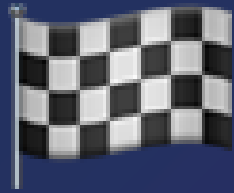
ESCAPE ROOM

3 AI agents, JSON
actions, pixel sprites

NPC CIVILIZATION

Phaser.js game with
AI-driven NPCs

All of these run **100% in the browser**. All included in the repo.



PHASE 4

Ship It

DEPLOY YOUR AGENT

1. Create a new repo on GitHub
2. Push your HTML file into it, named `index.html`
3. Go to **Settings** → **Pages** → set source to the main branch → Save



Your agent is **live** at
`username.github.io/repo-name`

RESOURCES + WHAT'S NEXT

WebLLM docs

github.com/mlc-ai/web-llm

MLC model zoo

huggingface.co/mlc-ai

Gemma on HF

huggingface.co/google/gemma-2-2b-it

 **All code**

github.com/nkthakur48/webllm-workshop

Go deeper:

 **Web Workers** - off main thread

 **Structured JSON** - valid JSON output

 **Bigger models** - Gemma 9B, 27B

 **Function calling** - native tool-calls



FEEDBACK

Tell me what worked, what didn't, and what you want next 😊



Scan to give feedback



Nishant Thakur

[in](#) /in/nishantkthakur

[🐙](#) /nkthakur48

