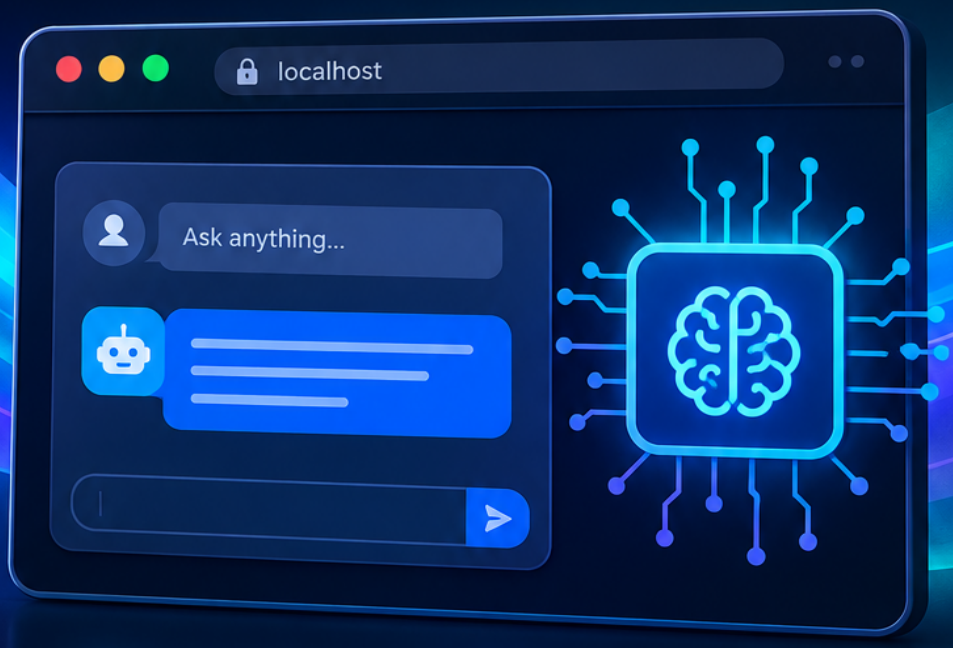
AI in the Browser

Building Privacy-Preserving Local Intelligence with WebGPU, WebAssembly, and On-Device LLMs



WebGPU
Compute Shaders

WebAssembly
High Performance

On-Device LLMs
Private & Local

Inference • Quantization • KV Caching • Streaming • Offline Deployment

DAVID L. HARPER

AI in the Browser

Building Privacy-Preserving Local Intelligence with WebGPU, WebAssembly, and On-Device LLMs

Steve Publications

This book is available at https://leanpub.com/aiinthebrowser

This version was published on 2026-09-03

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

Building Privacy-Preserving Local Intelligence with WebGPU, WebAssembly, and On-Device LLMs

About this book

The browser has become a viable platform for running sophisticated artificial intelligence models locally. This book takes you from foundational concepts through production-ready implementations of client-side AI applications that run entirely in the user's browser without requiring server-side inference backends. You will learn WebGPU compute shaders, WebAssembly integration, transformer architecture for inference engineers, model quantization, KV caching, streaming generation and deployment patterns for offline-capable AI web applications. Every chapter includes complete, runnable code examples that build progressively from GPU-accelerated matrix multiplication to a polished local LLM chat interface.

Introduction: The Browser as an AI Platform

In late 2025 something shifted quietly but profoundly. WebGPU became available by default across all major browsers. Chrome had supported it for years, but when Firefox 141 and Safari 26 shipped with full implementations, the story changed from "this is experimental" to "this is now part of the platform." The implication was immediate: any web page could now access GPU compute directly, using a standardized API that works across vendors and operating systems.

That means you can run a large language model inside someone's browser tab. Not an API call to OpenAI or Anthropic. Not even a request to your own server with CUDA-accelerated GPUs. The actual neural network inference happens on the user's device, using their GPU, entirely offline after the initial model download. Their prompts never leave their machine. Their data stays private by default because there is nowhere for it to go.

This book is about how that works and how to build real applications using it.

Why This Matters

The dominant paradigm for AI in web applications has been simple: user sends text to a server, the server runs inference on powerful GPUs, and streams tokens back. It works well enough but carries inherent limitations. Every token costs money. Every request adds latency. Every prompt traverses networks and touches infrastructure you may not fully control. Privacy is achieved through promises and encryption rather than architecture.

Client-side inference changes the trade-offs fundamentally. You trade compute cost for bandwidth: instead of paying per token, you pay once to deliver a model file that might be several gigabytes. After that initial download, inference is free. Latency drops because there is no network round-trip

between tokens. Privacy becomes structural rather than contractual since data never needs to leave the device.

This is not just academic. Real applications are shipping with this pattern now:

- A code editor extension that runs a reasoning model locally to explain and generate code, with full access to the DOM but zero server calls
- Medical documentation tools where patient data must never leave the hospital network, using local models for drafting notes
- Offline-capable translation apps that work on planes, in remote areas, or in networks with strict firewalls
- Educational tools that adapt to student writing patterns without sending their essays to a cloud provider

The technology stack enabling this is threefold: WebGPU provides direct GPU compute access from JavaScript, WebAssembly delivers near-native CPU performance for numerical code, and the browser itself supplies the runtime, memory management, storage APIs, and concurrency model that tie everything together.

What You Will Build

This book takes a progressive approach. We start with fundamentals and build toward complete applications. By the end you will have constructed:

1. A GPU-accelerated matrix multiplication kernel using WebGPU compute shaders and WGSL
2. A tensor operations library supporting the core operations needed for neural network inference
3. A WebAssembly module compiled from C/C++ that performs numerical computation alongside your JavaScript code
4. A complete local LLM inference engine that loads a GGUF model, runs transformer layers on the GPU, and generates text token by token
5. A streaming chat interface with conversation history management running in a Web Worker to keep the UI responsive
6. An offline-capable progressive web application with model caching, install prompts, and graceful fallbacks

Each project builds on the previous ones. You will see the same concepts recur at increasing levels of sophistication: memory management starts with simple buffer uploads and ends with KV cache allocation for multi-gigabyte models. Concurrency begins with basic Web Workers and culminates in a production-ready inference pipeline that streams tokens to the UI while computation continues unblocked.

Prerequisites and Approach

This book assumes you are comfortable with modern JavaScript or TypeScript, including ES modules, async/await and typed arrays like Float32Array. You should understand how browsers work at a basic level: the event loop, DOM manipulation, fetch API and service workers. Prior machine learning experience is not required but will help with some chapters.

The code examples use modern web tooling where it matters (TypeScript for larger projects, Vite or esbuild for bundling) but avoid unnecessary abstraction layers. When we write a WebGPU compute shader, you will see the complete WGSL source and the JavaScript that compiles and dispatches it. When we load a model from GGUF format, you will see the parsing logic that reads the header, extracts metadata, and maps tensors into GPU memory.

I prioritize clarity over cleverness. The goal is for you to understand what is happening at each layer so you can adapt these patterns to your own projects, debug problems when they arise, and make informed trade-offs about architecture and performance.

How to Use This Book

Read the chapters in order if you are new to browser-based AI. Each chapter depends on concepts introduced earlier: WebGPU fundamentals before shader programming for neural networks, transformer architecture before building an inference engine, KV caching before streaming generation.

If you already know some of this material, use it as a reference. The code examples are self-contained enough that you can jump to specific chapters and run the projects independently. Every major implementation includes the complete project structure, all source files, build commands and expected behavior so you can verify things work on your system.

Some APIs discussed in this book are still evolving. WebGPU is stable across major browsers as of late 2025 but may require feature flags or specific browser versions on some platforms. WebNN (Web Neural Network API) remains experimental. I note the stability status and browser support for each technology where it matters so you can make informed decisions about production use.

Now let us begin with understanding the browser as a platform for AI workloads.

Chapter 1: The Browser as an AI Platform

Why Local AI Matters

The shift toward client-side inference is driven by four concrete advantages that matter to both users and developers.

Privacy by architecture. When inference runs on the server, privacy depends on trust: you must believe the provider will not log prompts, train on your data without consent, or leak information through side channels. With local inference, the model runs in the browser sandbox on the user's own device. Prompts never leave the machine unless your application explicitly sends them somewhere. This is not a policy choice; it is a structural property of the system. For applications handling sensitive data (medical notes, legal documents, personal journals), this distinction is fundamental rather than incremental.

Latency without round trips. Server-based generation incurs at least one network round-trip per request, plus queuing time on shared infrastructure. Local inference eliminates both. Once the model is loaded and warmed up, token generation happens entirely on-device. On modern hardware with WebGPU acceleration, this means sub-100-millisecond time-to-first-token for small models and interactive streaming speeds (20 to 80 tokens per second depending on model size and device). The user experience feels instantaneous because there is no waiting for a server response.

Offline capability. A local model works without an internet connection after the initial download. This matters in environments where connectivity is unreliable (field work, remote locations), restricted (air-gapped networks, strict firewalls), or expensive (mobile data on limited plans). Progressive web applications can cache models using service workers and IndexedDB so that users install the app once and use it indefinitely offline.

Cost predictability. Server inference costs scale with usage: every token generated has a price tag. For high-volume applications this becomes signif-

icant quickly. Local inference flips the cost model: you pay for bandwidth to deliver the model file (a one-time cost per user), then inference is free. This makes budgeting predictable and can dramatically reduce operational expenses for applications where many users generate substantial amounts of text.

These advantages are not theoretical. They drive real architectural decisions in production systems built today.

What the Browser Can Actually Do

Before building anything, it is important to understand realistic capabilities and constraints. The browser is powerful but not unlimited.

Model sizes that fit. A browser tab's memory budget depends on the system but typically ranges from 2GB to 8GB+ on modern desktops with sufficient RAM. This constrains model size:

- 1B parameter models (quantized): ~600MB to 1.5GB : runs comfortably on most devices
- 3B parameter models (quantized): ~1.5GB to 2.5GB : viable on modern laptops and desktops
- 7B parameter models (quantized Q4_K_M): ~4GB : requires decent hardware, works well with WebGPU on capable GPUs
- 8B parameter models (quantized Q4_K_M): ~5GB : pushes limits but runs on mid-range to high-end devices

These numbers assume quantization to 4-bit or similar precision. Full FP16 models require roughly twice the memory and are rarely practical in browsers except for very small architectures. The trade-off between model size and quality is real: a 1B parameter model cannot match a 70B parameter model's reasoning ability, but it can handle many practical tasks (summarization, translation, code assistance, drafting) at speeds that feel excellent.

Inference speeds on typical hardware. Performance varies dramatically based on device capability and whether WebGPU acceleration is available:

- MacBook Pro M3 Max with WebGPU: 60 to 80+ tokens per second for Phi-3.5-mini (3.8B parameters)

- Desktop NVIDIA RTX 4070 with WebGPU: 40 to 60 tokens per second for Llama-3.1-8B Q4_K_M
- Mid-range laptop GPU with WebGPU: 20 to 40 tokens per second for 3B parameter models
- CPU-only via WebAssembly: 5 to 15 tokens per second for small models, often too slow for comfortable interaction

The gap between WebGPU and WebAssembly is substantial. WebGPU provides 10x to 20x speedups for transformer workloads on capable hardware because matrix multiplication dominates inference time and GPUs execute those operations in parallel at scale. WebAssembly with SIMD extensions helps but cannot match GPU throughput for the heavy tensor operations transformers require.

Memory constraints. The browser imposes hard limits that desktop applications do not face:

- Per-tab memory limits (typically 1GB to 4GB depending on system RAM and browser)
- Garbage collection pauses when JavaScript heap pressure is high
- No direct file system access without user permission (though Origin Private File System helps)
- Shared resources across tabs from the same origin

These constraints mean you cannot simply load a 70B parameter model into a browser tab the way you might on a server with 128GB of RAM. You must work within realistic bounds and design applications that respect them.

The Three Pillars

Three technologies converge to make browser-based AI viable: WebGPU, WebAssembly and the browser runtime itself.

WebGPU for parallel compute. WebGPU is a low-level graphics and compute API standardized by W3C's GPU for the Web community group. Unlike its predecessor WebGL (which was designed primarily for rendering triangles), WebGPU exposes general-purpose GPU computation through compute shaders written in WGSL (WebGPU Shading Language). This matters because

neural network inference is fundamentally parallel linear algebra: thousands of matrix multiplications that map naturally to GPU execution units.

WebGPU provides explicit memory management (no hidden allocations), fine-grained control over GPU pipelines and direct access to modern GPU features like shared memory within workgroups and hardware-accelerated tensor operations on supported devices. It abstracts away vendor-specific APIs (Vulkan, Metal, Direct3D) into a single cross-platform interface.

WebAssembly for portable performance. WebAssembly is a binary instruction format that runs in the browser alongside JavaScript at near-native speed. For AI workloads it serves two purposes: first as a fallback when WebGPU is unavailable (compiled inference engines like llama.cpp run acceptably on CPU via Wasm), and second for operations that do not map cleanly to GPU shaders (tokenization, certain preprocessing steps, control logic).

WebAssembly integrates with JavaScript through typed arrays that share memory regions. Code compiled from C/C++ or Rust runs in a sandboxed environment with deterministic performance characteristics. SIMD extensions enable vectorized computation on CPU, which helps for smaller models or operations that cannot be parallelized across GPU threads efficiently.

The browser runtime as integration layer. The browser itself provides critical infrastructure: the JavaScript engine (V8, SpiderMonkey, JavaScriptCore) orchestrates everything; Web Workers provide concurrency without blocking the UI thread; IndexedDB and Cache API store multi-gigabyte models persistently; service workers enable offline functionality; and the security model sandboxing ensures that even powerful local computation stays contained.

These three layers work together: JavaScript coordinates the application logic, WebGPU handles heavy parallel tensor operations, WebAssembly runs specialized numerical kernels or CPU fallbacks and browser APIs manage storage, concurrency and user interaction. Understanding how they integrate is essential for building performant systems.

A Working Example

Before diving into implementation details, let us look at what a complete local LLM application looks like in practice. This example uses Transformers.js, a library that abstracts much of the complexity but demonstrates the architecture end-to-end.

Create a file named `index.html`:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Local LLM Chat</title>
    <style>
        * { box-sizing: border-box; margin: 0; padding: 0; }
        body { font-family: system-ui, -apple-system, sans-serif; max-width: 800px; margin: 0 auto; padding: 20px; background: #f5f5f5; }
        #status { padding: 12px; border-radius: 8px; margin-bottom: 16px; font-size: 14px; }
        #status.loading { background: #fff3cd; color: #856404; }
        #status.ready { background: #d4edda; color: #155724; }
        #chat { background: white; border-radius: 12px; padding: 20px; min-height: 400px; box-shadow: 0 2px 8px rgba(0,0,0,0.1); margin-bottom: 16px; }
        .message { margin-bottom: 16px; padding: 12px; border-radius: 8px; }
        .user { background: #e3f2fd; margin-left: 40px; }
        .assistant { background: #f5f5f5; margin-right: 40px; }
        .role { font-weight: 600; font-size: 12px; text-transform: uppercase; color: #666; margin-bottom: 4px; }
        #input-area { display: flex; gap: 8px; }
        #prompt { flex: 1; padding: 12px; border: 1px solid #ddd; border-radius: 8px; font-size: 16px; resize: none; height: 50px; }
        #send { padding: 12px 24px; background: #1976d2; color: white; border: none; border-radius: 8px; cursor: pointer; font-size: 16px; }
        #send:disabled { background: #ccc; cursor: not-allowed; }
        #abort { padding: 12px 24px; background: #d32f2f; color: white; border: none; border-radius: 8px; cursor: pointer; font-size: 16px; display: none; }
    </style>
</head>
<body>
    <h1 style="margin-bottom: 16px;">Local LLM Chat</h1>
    <div id="status" class="loading">Initializing WebGPU...</div>
    <div id="chat"></div>
    <div id="input-area">
        <textarea id="prompt" placeholder="Type your message..." disabled></textarea>
        <button id="send" disabled>Send</button>
        <button id="abort">Stop</button>
    </div>

    <script type="module">
        import { pipeline, env } from 'https://cdn.jsdelivr.net/npm/@huggingface/transformers@3.0.0';

```

```
        // Disable local model loading; use CDN only
        env.localHubUrl = null;

        const statusEl = document.getElementById('status');
        const chatEl = document.getElementById('chat');
        const promptEl = document.getElementById('prompt');
        const sendBtn = document.getElementById('send');
        const abortBtn = document.getElementById('abort');

        let pipe = null;
        let abortController = null;
        let conversationHistory = [];

        // Check WebGPU support
        async function checkWebGPU() {
            if (!navigator.gpu) {
                statusEl.textContent = 'WebGPU not supported. Falling back to
                ↪  CPU (slow).';
                statusEl.className = 'loading';
                return false;
            }
            try {
                const adapter = await navigator.gpu.requestAdapter();
                if (!adapter) {
                    statusEl.textContent = 'No GPU adapter found. Falling back
                    ↪  to CPU.';
                    return false;
                }
                return true;
            } catch (e) {
                console.error('WebGPU error:', e);
                return false;
            }
        }

        // Load the model
        async function loadModel() {
            const hasWebGPU = await checkWebGPU();
            const device = hasWebGPU ? 'webgpu' : 'wasm';
            statusEl.textContent = `Loading model on ${device.toUpperCase()}...
            ↪  This may take a minute.`;

            try {
                // Use a small model optimized for browser inference
                pipe = await pipeline(
                    'text-generation',
                    'Xenova/Llama-3.2-1B-Instruct',
                    { device }
                );
```

```
84                  statusEl.textContent = `Ready! Running on
                    ↪  ${device.toUpperCase()}. Your messages never leave this
                    ↪  browser.`;
85                  statusEl.className = 'ready';
86                  promptEl.disabled = false;
87                  sendBtn.disabled = false;
88              } catch (error) {
89                 statusEl.textContent = `Failed to load model: ${error.message}`;
90                  console.error(error);
91              }
92          }
93
94          // Add message to chat display
95          function addMessage(role, content) {
96              const msgDiv = document.createElement('div');
97              msgDiv.className = `message ${role}`;
98              msgDiv.innerHTML = `<div class="role">${role}</div><div
                ↪  class="content">${escapeHtml(content)}</div>`;
99              chatEl.appendChild(msgDiv);
100             chatEl.scrollTop = chatEl.scrollHeight;
101             return msgDiv;
102         }
103
104         // Escape HTML to prevent XSS
105         function escapeHtml(text) {
106             const div = document.createElement('div');
107             div.textContent = text;
108             return div.innerHTML;
109         }
110
111         // Generate response with streaming
112         async function generate() {
113             const userMessage = promptEl.value.trim();
114             if (!userMessage || !pipe) return;
115
116             abortController = new AbortController();
117             promptEl.value = '';
118             promptEl.disabled = true;
119             sendBtn.disabled = true;
120             abortBtn.style.display = 'inline-block';
121
122             addMessage('user', userMessage);
123
124             // Build conversation history
125             conversationHistory.push({ role: 'user', content: userMessage });
126             const messages = conversationHistory.map(msg => `${msg.role}:
                ↪  ${msg.content}`);
127             const prompt = pipe.chat_template.apply_chat_template(messages, {
128                 add_generation_prompt: true,
129                 tokenize: false
```

```
        });

        const assistantMsg = addMessage('assistant', '');
        let fullResponse = '';

        try {
            const output = await pipe(prompt, {
                max_new_tokens: 256,
                temperature: 0.7,
                top_p: 0.9,
                do_sample: true,
                return_full_text: false,
                // Stream tokens as they are generated
                callback_function: (token) => {
                    if (abortController?.signal.aborted) {
                        throw new Error('Generation aborted');
                    }
                    const text = pipe.tokenizer.decode([token.id], {
                    ↪  skip_special_tokens: true });
                    fullResponse += text;
                    assistantMsg.querySelector('.content').textContent =
                    ↪  fullResponse;
                    chatEl.scrollTop = chatEl.scrollHeight;
                }
            });

            conversationHistory.push({ role: 'assistant', content:
            ↪  fullResponse });
        } catch (error) {
            if (error.message !== 'Generation aborted') {
                console.error('Generation error:', error);
                assistantMsg.querySelector('.content').textContent =
                    fullResponse + '[Error during generation]';
            }
        } finally {
            promptEl.disabled = false;
            sendBtn.disabled = false;
            abortBtn.style.display = 'none';
            abortController = null;
            promptEl.focus();
        }
    }

    // Event listeners
    sendBtn.addEventListener('click', generate);
    promptEl.addEventListener('keydown', (e) => {
        if (e.key === 'Enter' && !e.shiftKey) {
            e.preventDefault();
            generate();
        }
```

```
        });
        abortBtn.addEventListener('click', () => {
            abortController?.abort();
        });

        // Initialize
        loadModel();
    </script>
</body>
</html>
```

Open this file in a modern browser (Chrome, Edge, Firefox 141+, or Safari 26+). On first load it will download the Llama-3.2-1B-Instruct model from Hugging Face's CDN : approximately 900MB for the quantized version. This happens once; subsequent loads use the browser cache. After loading completes, you can chat with the model entirely offline. Your messages and responses never leave your browser.

This example uses Transformers.js to abstract away the complexity of WebGPU setup, GGUF parsing, tensor operations, and transformer layer execution. The rest of this book explains what happens underneath that abstraction so you can build custom solutions, optimize for specific use cases, and debug problems when they arise.

What This Book Will Build

We will construct increasingly sophisticated applications across twelve chapters:

1. **The Browser as an AI Platform** (this chapter): Understanding capabilities, constraints, and the technology stack
2. **Browser Architecture and AI Workloads**: How the browser executes code, manages memory, and handles concurrency for compute-heavy tasks
3. **WebGPU Fundamentals**: Device initialization, buffer management, compute pipelines from scratch
4. **Shader Programming for Neural Networks**: Writing WGSL compute shaders for tensor operations used in inference
5. **WebAssembly Fundamentals and Toolchains**: Compiling C/C++ and Rust to Wasm for numerical computation

6. **Interoperability**: Integrating JavaScript, WebAssembly, and WebGPU into a coherent system
7. **Transformer Architecture for Inference Engineers**: Understanding transformers from the perspective of implementing inference
8. **Model Formats, Quantization and Loading**: GGUF format, quantization schemes, streaming model loads
9. **Building a Local LLM Inference Engine**: Complete implementation of an inference pipeline from scratch
10. **Streaming Generation and User Experience**: Web Workers, streaming tokens, conversation management
11. **Performance Optimization and Debugging**: Profiling, optimization techniques, cross-browser compatibility
12. **Production Deployment and Future Directions**: PWA deployment, offline capability, security, emerging APIs

Each chapter's code examples are designed to be runnable independently but also reference concepts from earlier chapters. By the final chapter you will have a complete understanding of how browser-based AI works at every layer, from GPU shader execution to user-facing chat interfaces.

Summary

The browser is now a legitimate platform for running AI models locally. WebGPU provides GPU compute access, WebAssembly delivers CPU performance and browser APIs handle storage, concurrency and offline capability. The trade-offs are clear: you exchange per-token server costs for upfront model download bandwidth, but gain privacy, lower latency, offline operation, and predictable pricing.

The technology is mature enough for production use on modern browsers (all major browsers support WebGPU as of late 2025), though you must design within realistic constraints around model size, memory limits, and hardware variability. The next chapter examines browser architecture in detail so we understand the execution environment before writing GPU code.

Chapter 2: Browser Architecture and AI Workloads

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

The Browser Engine

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Memory Management in the Browser

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Concurrency and Web Workers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Storage for Large Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Browser Security Model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Chapter 3: WebGPU Fundamentals

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Why WebGPU

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Device Initialization and Adapter Selection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Buffers and Memory Layout

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Compute Pipelines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Building a GPU-Accelerated Matrix Multiplication

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Chapter 4: Shader Programming for Neural Networks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

The Compute Shader Model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

WGSL Basics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Vector Operations in Shaders

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Matrix Multiplication Shader

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Tensor Operations for Transformers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Chapter 5: WebAssembly Fundamentals and Toolchains

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

What WebAssembly Is and Is Not

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Compiling C/C++ with Emscripten

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Rust for WebAssembly

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Wasm Memory and Typed Arrays

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

A Numerical Kernel in Wasm

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Chapter 6: Interoperability : JavaScript, Wasm and WebGPU Together

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Data Flow Between Layers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Shared Buffers and Zero-Copy Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Calling Conventions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Architecture Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Building a Tensor Library Skeleton

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Chapter 7: Transformer Architecture for Inference Engineers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

The Forward Pass

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Self-Attention Mechanics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Key-Value Caching

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Layer Normalization Variants

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Positional Encodings

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Chapter 8: Model Formats, Quantization and Loading

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

GGUF Format

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Quantization Fundamentals

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Loading Models in the Browser

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Model Caching Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Choosing a Model for the Browser

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Chapter 9: Building a Local LLM Inference Engine

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Project Setup

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Tokenizer Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Model Loading and GPU Buffer Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

WebGPU Compute Kernels

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

The Inference Loop

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Sampling Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Main Application Bootstrap

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Expected Behavior and Running the Project

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Chapter 10: Streaming Generation and User Experience

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Streaming Tokens to the UI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Web Workers for Inference

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Conversation State Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Abort and Control Flow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Building a Chat Interface

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Chapter 11: Performance Optimization and Debugging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Profiling Browser AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Memory Optimization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

GPU Optimization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Latency vs Throughput

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Cross-Browser and Hardware Compatibility

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Chapter 12: Production Deployment and Future Directions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Offline AI PWAs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Progressive Enhancement

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Privacy and Security

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Packaging and Distribution

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

The Future of Browser AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

Conclusion: The Local Intelligence Paradigm

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/aiinthebrowser.