LiteRT.js

High-Performance On-Device AI Inference in the Browser

The Definitive Guide to Running Machine Learning Models Locally with WebAssembly, WebGPU, and WebNN

"Bring the power of machine learning to your users—no servers required."
– Run AI. Anywhere.

PATRICK MACIEJ

LiteRT.js: High-Performance On-Device AI Inference in the Browser

The Definitive Guide to Running Machine Learning Models Locally with WebAssembly, WebGPU, and WebNN

Steve T. Publications

This book is available at https://leanpub.com/litertjshigh-performanceon-deviceaiinferenceinthebrowser

This version was published on 2026-07-13

Contents

The Definitive Guide to Running Machine Learning Models Locally with WebAssembly, WebGPU, and WebNN

About this book: LiteRT.js represents a paradigm shift in web-based artificial intelligence. By bringing Google's native, cross-platform LiteRT runtime into the browser through WebAssembly, it enables production-grade, hardware-accelerated model inference without any server dependencies. This book provides the complete technical foundation and practical implementation patterns for intermediate to advanced JavaScript developers who want to run machine learning models directly in the user's browser. From understanding the architecture and mastering the API to building real-world applications in image classification, object detection, speech recognition, LLM chatbots, and beyond, every chapter builds toward production-ready implementations with detailed code examples, performance optimization strategies, and deployment guidance.

Introduction

A few years ago, running a neural network in a web browser meant accepting significant compromises. You could send user data to a cloud server and wait for the response, paying per inference while sacrificing privacy. Or you could use a JavaScript-based inference library and accept that complex models would crawl at a fraction of their native speed. The browser was never designed to be an AI platform, and the tools reflected that limitation.

That changed with LiteRT.js.

Announced in July 2026 by Google, LiteRT.js is not merely another JavaScript machine learning library. It is a fundamental rethinking of how on-device AI runs in the browser. Instead of implementing neural network operators in JavaScript, LiteRT.js compiles Google's production-grade LiteRT runtime to WebAssembly and exposes it through clean JavaScript bindings. The result is a runtime that delivers near-native performance while running entirely inside the user's browser, with full access to GPU acceleration through WebGPU and emerging NPU support through WebNN.

The implications are substantial. A financial services application can perform OCR on uploaded documents without ever transmitting sensitive data to a server. A medical imaging tool can run segmentation models on patient scans entirely client-side, satisfying HIPAA requirements by design. An offline-first field application for agricultural inspection works in remote areas with no network connectivity. A chatbot powered by an on-device large language model generates responses with zero latency from network round-trips and complete user privacy.

These are not hypothetical scenarios. They represent the class of applications that LiteRT.js makes practical today.

What This Book Covers

This book is designed for developers who already understand JavaScript, modern browser APIs, and the basics of machine learning, and who want to

build production-quality AI applications that run entirely in the user's browser. It assumes you are comfortable with package managers, bundlers, and modern web frameworks, and it expects you to write real code.

The book progresses from foundational concepts to advanced implementations:

- **Architecture and internals**: You will understand exactly how LiteRT.js bridges JavaScript to WebAssembly, manages tensor memory, delegates work to hardware accelerators, and compiles models for execution.
- **Complete API mastery**: Every function, class, configuration option, and parameter is covered with detailed explanations and working code examples.
- **Hardware acceleration**: You will learn when and how to use WebGPU, WebNN, and the XNNPACK CPU backend, including the trade-offs between each approach.
- **Model conversion and optimization**: From converting PyTorch, TensorFlow, and JAX models to the .tflite format, through quantization strategies that reduce model size by orders of magnitude while preserving accuracy.
- **Framework integration**: Production patterns for React, Vue, Angular, Svelte, Next.js, and Vite, including build configuration for serving WebAssembly binaries.
- **Real-world applications**: Complete implementations of image classification, object detection with YOLO, semantic segmentation, depth estimation, image upscaling, audio processing, speech recognition, text embeddings for RAG, OCR, and LLM chatbots using LiteRT-LM.
- **Production deployment**: Security considerations, browser compatibility, progressive enhancement, performance benchmarking, and migration from TensorFlow.js.

What This Book Is Not

This book does not teach machine learning fundamentals. If you need to understand what a neural network is, how backpropagation works, or how to train a model, there are excellent resources for that. This book assumes you can work with pre-trained models and focuses exclusively on the deployment and inference side of the equation.

This book also does not cover server-side model serving, training pipelines, or MLOps infrastructure. The entire scope is client-side, browser-based inference using LiteRT.js.

How to Use This Book

Read the chapters in order for the most complete understanding. Each chapter builds on concepts introduced earlier, and the code examples progressively increase in complexity. That said, if you are already familiar with basic WebAssembly and neural network inference concepts, you can jump directly to the chapters relevant to your use case.

Every code example is designed to be production-quality, not toy demonstrations. They include error handling, memory cleanup, and performance considerations that matter in real applications. When an alternative approach exists, the book explains the trade-offs so you can make informed decisions for your specific context.

Prerequisites

To get the most from this book, you should have:

- **Solid JavaScript/TypeScript skills**: Modern ES6+ syntax, async/await, modules, and typed arrays.
- **Understanding of browser APIs**: Familiarity with the Fetch API, Web Workers, and ideally some exposure to WebGL or Canvas.
- **Basic machine learning literacy**: Understanding what a neural network, tensor, and inference are, even if you have never trained a model yourself.
- **Package management experience**: Comfortable using npm or pnpm, and working with bundlers like Vite or webpack.
- **A modern browser**: Chrome 113+, Edge 113+, Firefox 141+, or Safari 26+ for full WebGPU support.

A Note on Versions

LiteRT.js is an active project. The npm package @litertjs/core is at version 2.5.x as of this writing, and the LiteRT-LM JavaScript API is in early preview. The APIs

described in this book reflect the latest stable releases, but you should always check the official documentation for the most current information. Breaking changes, while rare, can occur as the project matures, and the patterns and principles covered here will remain applicable even as specific API signatures evolve.

Let us begin.

Chapter 1: The Web AI Revolution – Why LiteRT.js Matters

The Browser as an AI Platform

The modern web browser is a remarkable piece of software. It runs complex applications, renders 3D graphics, processes video streams in real time, and manages millions of concurrent network connections. For years, it has been the de facto operating system for the internet. Yet until recently, one critical capability was conspicuously absent: high-performance machine learning inference.

That absence forced developers into a false dichotomy. Either send data to a cloud server for processing, accepting the latency, cost, and privacy implications, or attempt to run models in JavaScript using libraries that were never designed for production-grade performance. The gap between what browsers could theoretically do and what they actually delivered for AI workloads was enormous.

The architecture of modern browsers has evolved dramatically to close this gap. WebAssembly emerged as a portable binary format that runs at near-native speed inside the browser sandbox. WebGPU replaced the aging WebGL API with a modern, low-overhead interface to GPU compute capabilities. JavaScript Promise Integration (JSPI) bridged the synchronous-asynchronous boundary between WebAssembly and JavaScript. Together, these technologies created the foundation for running serious machine learning models directly in the browser.

What was missing was a runtime that brought all of these capabilities together with the kind of production-grade optimization that comes from years of cross-platform development. That is exactly what LiteRT.js delivers.

From TensorFlow.js to LiteRT.js: A Generational Shift

To understand why LiteRT.js matters, it helps to understand what came before. TensorFlow.js was Google's first major effort to bring machine learning inference to the browser. It was a pioneering project that demonstrated the viability of client-side ML and built a substantial ecosystem around it. But its architecture had fundamental limitations.

TensorFlow.js implements neural network operators in JavaScript, with optional GPU acceleration through WebGL shaders. This approach means every matrix multiplication, convolution, and activation function runs through layers of JavaScript interpretation overhead. The WebGL backend adds additional complexity because graphics APIs were never designed for general-purpose compute workloads. The result is that even on powerful hardware, TensorFlow.js typically delivers inference speeds that are a small fraction of what the underlying hardware is capable of.

The model conversion pipeline for TensorFlow.js was equally painful, particularly for PyTorch models. Converting a PyTorch model to TensorFlow.js required converting to ONNX first, then from ONNX to TensorFlow SavedModel format, and finally to the TensorFlow.js Graph Model format. Each step introduced potential compatibility issues, operator support gaps, and shape mismatches that could take hours or days to debug.

LiteRT.js addresses both of these problems at the architectural level. Instead of implementing operators in JavaScript, it compiles Google's LiteRT runtime – a C++ inference engine optimized across Android, iOS, desktop, and embedded platforms – to WebAssembly. This means the same highly optimized kernels that power on-device AI in billions of mobile devices now run inside your browser. The performance difference is dramatic: Google's benchmarks show up to three times faster inference compared to competing web solutions on both CPU and GPU backends.

The conversion pipeline is equally simplified. PyTorch models convert directly to the .tflite format using the litert_torch converter in a single step, bypassing the fragile ONNX intermediary entirely. The same .tflite model file runs across Android, iOS, desktop, and the web with zero modifications.

What Is LiteRT? The Unified On-Device Runtime

LiteRT (pronounced "light runtime") is Google's unified on-device machine learning framework, succeeding TensorFlow Lite. It represents a fundamental shift in how Google approaches on-device AI: rather than maintaining separate runtimes for different platforms, LiteRT provides a single core runtime with platform-specific accelerators.

The LiteRT architecture consists of three layers:

1. **Model format**: The .tflite FlatBuffer format encodes the computation graph, tensor shapes, data types, and operator specifications in a compact binary representation. This format is universal across all platforms and LiteRT versions.
2. **Core runtime**: Written primarily in C++, the core runtime handles model loading, graph compilation, memory management, and execution scheduling. It is platform-agnostic and shared across all deployment targets.
3. **Accelerator backends**: Platform-specific delegates handle the actual computation. On mobile, this includes GPU delegates for Qualcomm Adreno, ARM Mali, and Apple GPUs, plus NPU delegates for Google Tensor and Samsung Exynos chips. On desktop, OpenCL, OpenGL, Metal, and Vulkan backends provide GPU acceleration. In the browser, WebAssembly with XNNPACK handles CPU execution, while WebGPU and WebNN provide GPU and NPU access respectively.

This architecture means that optimizations developed for one platform automatically benefit all others. A quantization improvement discovered on Android improves web inference. A new operator implementation for the GPU delegate becomes available across mobile and desktop simultaneously. The cross-platform nature of LiteRT is not a marketing claim; it is an architectural guarantee.

LiteRT.js extends this unified stack to the web by compiling the core runtime to WebAssembly and implementing browser-specific accelerator delegates. The JavaScript API provides idiomatic access to the underlying C++ runtime while managing the complexities of WebAssembly memory, GPU buffer transfers, and asynchronous execution.

The Value Proposition: Privacy, Performance, and Cost

The decision to run AI inference in the browser rather than on a server comes down to three factors: privacy, performance, and cost. LiteRT.js excels at all three.

Privacy. When inference runs entirely in the browser, user data never leaves the device. This is not an incremental improvement over server-side processing; it is a fundamental architectural difference. For applications handling sensitive data – medical images, financial documents, personal photographs, private communications – this distinction has legal and regulatory significance. GDPR compliance becomes simpler when there is no data transfer to document. HIPAA requirements are satisfied by design rather than through additional engineering effort.

Consider a document processing application for a law firm. With server-side inference, every uploaded document must be transmitted to a cloud service, processed, and the results returned. Even with encryption in transit, the data exists on servers outside the firm's control. With LiteRT.js, OCR models run entirely in the browser. The document never leaves the user's device. Personally identifiable information can be detected and redacted locally before any server upload occurs.

Performance. Latency is the most visible performance metric for end users, but it is not the only one. Server-side inference introduces network round-trip latency that varies based on user location, network conditions, and server load. Even a fast 50 millisecond inference becomes 200 milliseconds or more when network latency is included. Browser-based inference eliminates this variable entirely: the model runs on the device the user is already holding.

Throughput matters too. Server-side processing requires scaling infrastructure to handle peak loads, which means paying for capacity that sits idle during off-peak hours. Browser-based inference distributes the computational load across all users simultaneously. Adding one hundred more users adds zero server cost because each user's device handles their own inference.

Cost. The economics of client-side AI are straightforward: every inference that runs in the browser is an inference that does not run on your server. For applications with high inference volumes, the cost savings are substantial. A model that costs $0.001 per inference on a cloud GPU service costs nothing when run client-side, beyond the one-time cost of downloading the model file

to the user's device.

This does not mean browser-based inference replaces server-side processing entirely. Some models are too large to fit in browser memory. Some workloads require access to server-side data stores or other backend services. But for a significant class of applications – image classification, object detection, text classification, audio processing, and even lightweight language models – the browser is now a viable and often superior inference platform.

When to Use (and Not Use) LiteRT.js

LiteRT.js is powerful, but it is not universally applicable. Understanding its boundaries prevents wasted effort and ensures that you choose the right tool for each situation.

Use LiteRT.js when:

- Your model fits within browser memory constraints (typically under 500 MB for the model file, though this varies by device).
- The inference latency requirements can be met by client hardware (modern devices with capable GPUs typically handle image classification in under 100 milliseconds).
- Privacy or data locality is a requirement rather than a nice-to-have.
- You need to reduce server infrastructure costs for high-volume inference workloads.
- Offline operation is required or desirable.
- Your target browsers support WebAssembly and preferably WebGPU (Chrome 113+, Edge 113+, Firefox 141+, Safari 26+).

Consider alternatives when:

- Your model exceeds browser memory limits. Large transformer models, especially those with billions of parameters, may not be practical for browser deployment without aggressive quantization.
- You need to process extremely large inputs (high-resolution video streams, massive point clouds) that would overwhelm client hardware.
- Your target audience uses older browsers without WebAssembly or WebGPU support and you cannot provide a fallback experience.

- The inference requires access to server-side data, external APIs, or other backend services that cannot be replicated client-side.
- You need to run custom operators that are not supported by any of the LiteRT.js accelerator backends.

The boundary between these categories is shifting as browser capabilities improve and models become more efficient. A model that was impractical for browser deployment six months ago may be perfectly viable today thanks to improved WebGPU support and better quantization techniques. Stay current with browser compatibility data and LiteRT.js release notes to make informed decisions for your specific use case.

The Landscape of Browser AI

Before committing to LiteRT.js, it is worth understanding where it fits in the broader ecosystem of browser-based machine learning. Several alternative approaches exist, each with their own strengths and weaknesses.

TensorFlow.js remains the most mature browser ML library by age, with a large community and extensive documentation. Its WebGL backend provides GPU acceleration, though at lower performance levels than WebGPU-based solutions. The interoperability package @litertjs/tfjs-interop makes it possible to use LiteRT.js within existing TensorFlow.js pipelines, allowing gradual migration rather than a complete rewrite.

ONNX Runtime Web supports the Open Neural Network Exchange format and provides both WebAssembly and WebGPU backends. It is framework-agnostic, accepting models from PyTorch, scikit-learn, and other frameworks that export to ONNX. Its operator coverage is broad, though performance varies by model type.

Transformers.js, built on Hugging Face's transformers library, specializes in natural language processing models. It uses ONNX Runtime Web under the hood and provides a familiar API for developers working with text classification, tokenization, and sequence-to-sequence tasks.

WebLLM focuses specifically on large language model inference in the browser, using WebGPU compute shaders for high-performance execution. It supports models like Llama, Mistral, and Phi with quantized weights.

LiteRT.js differentiates itself through its direct connection to Google's production-grade LiteRT runtime. The same optimization pipeline that powers AI features in Chrome, Android, and Pixel devices flows directly into the web runtime. This means browser applications automatically benefit from performance improvements, new operator support, and quantization advances developed for billions of mobile devices worldwide.

In the chapters that follow, we will explore every aspect of LiteRT.js in depth. You will learn how it works internally, how to use every API, how to optimize models for maximum performance, and how to build production applications across a wide range of use cases. The foundation is laid; let us now examine the architecture.

Chapter 2: Architecture Deep Dive

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

The JS-to-WebAssembly Bridge Pattern

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

Model Loading and Compilation Pipeline

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

Tensor Buffer Management and Memory Layout

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

Accelerator Delegation System

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

SignatureRunner and Named Entry Points

Chapter 3: Getting Started – Installation, Initialization, and First Inference

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

Installing @litertjs/core

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

Serving WebAssembly Binaries

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

Runtime Initialization with loadLiteRt

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

Loading and Compiling Your First Model

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

Running Inference and Reading Results

Chapter 4: Hardware Acceleration – WebGPU, WebNN, and XNNPACK

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

WebGPU: GPU-Accelerated Inference

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

WebNN: Neural Processing Unit Access

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

XNNPack on WebAssembly: Optimized CPU Execution

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

Accelerator Selection Strategy and Fallback Patterns

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

JSPI (JavaScript Promise Integration) for Asynchronous Execution

Chapter 5: Model Conversion – From PyTorch, TensorFlow, and JAX to .tflite

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

The .tflite FlatBuffer Format

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

Converting PyTorch Models with litert_torch

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

Converting TensorFlow and JAX Models

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

Ultralytics YOLO Export Integration

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

Conversion Troubleshooting and Common Pitfalls

Chapter 6: Model Optimization – Quantization, Size Reduction, and Performance Tuning

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

Understanding Quantization: INT8, INT4, FP16

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

Dynamic, Static, and Weight-Only Quantization

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

Using the AI Edge Quantizer

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

Selective Quantization and Mixed-Precision Strategies

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

Model Size vs. Accuracy Trade-offs

Chapter 7: The Complete API Reference

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

Core Runtime Functions (loadLiteRt, loadAndCompile)

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

Tensor Class: Creation, Manipulation, and Transfer

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

CompiledModel and SignatureRunner APIs

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

Configuration Options and CompileOptions

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

Error Handling and Debugging Utilities

Chapter 8: Memory Management and Performance Engineering

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

Manual Memory Management: Why delete() Matters

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

GPU-to-CPU Data Transfer Patterns

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

Benchmarking Inference Performance

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

Profiling and Diagnosing Bottlenecks

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

Production Optimization Checklist

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

Chapter 9: Integration with Modern Web Frameworks

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

React Integration (react-litert and Custom Hooks)

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

Vue.js Composition API Integration

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

Angular Services and Components

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

Svelte Stores and Reactive Inference

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

Next.js and Vite Build Configuration for WASM

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

Chapter 10: Computer Vision Applications

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

Image Classification with MobileNet and ResNet

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

Real-Time Object Detection with YOLO

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

Semantic Segmentation Pipelines

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

Depth Estimation and 3D Point Clouds

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

Image Upscaling with Real-ESRGAN

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

Chapter 11: Audio, Speech, Text, and Multimodal AI

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

Audio Processing and Noise Suppression

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

Speech Recognition Patterns in the Browser

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

Text Embeddings and On-Device RAG

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

OCR and Document Intelligence

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

LLM Inference with LiteRT-LM.js

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

Chapter 12: Production Deployment – Security, Compatibility, and Best Practices

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

Security Model and Threat Considerations

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

Browser Compatibility and Feature Detection

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

Progressive Enhancement and Graceful Degradation

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

Migrating from TensorFlow.js

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

Production Deployment Checklist

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

Conclusion

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

References

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