CANONICAL LIVING OVERVIEW GUIDE

WebGPU, Browser-Local Compute, and High-Performance Web Systems

A WebGPU and browser-compute engineering guide covering adapters, devices, resources, WGSL, rendering, compute, synchronization, performance, browser-local AI, WebNN interoperability, security, privacy, debugging, deployment, and progressive enhancement.

PERMANENT ID
MYTHOS-FG-FRONTIER-0008

PUBLICATION CLASS
FIELD GUIDE

VERSION / STATUS
1.0.0-candidate / CANDIDATE

PERMANENT PUBLICATION IDENTITY

WebGPU, Browser-Local Compute, and High-Performance Web Systems

Document ID
MYTHOS-FG-FRONTIER-0008

Version
1.0.0-candidate

Status
Candidate Focused Field Guide

Review date
2026-10-26

Publication position
This is a living field guide. It is designed for frequent revision while retaining a stable permanent identity. Candidate status means the content is ready for structured use and review but is not represented as an external certification or authority publication.

LEVEL	CLASS	CADENCE	EVIDENCE
L1-L4	FIELD GUIDE	QUARTERLY	SOURCE-BOUND
Foundational through frontier	Practical and educational	Interim updates as needed	Limitations remain visible

Reader orientation

Dimension	Engineering position
Primary domain	MYTHOS-EMT - Emerging Technology & Sovereign Infrastructure
Secondary domain	MYTHOS-UX; MYTHOS-SWE; MYTHOS-AI
Audience	Web, graphics, AI, visualization, scientific-computing, browser-platform, and performance engineers.
Prerequisites	JavaScript or TypeScript, graphics fundamentals, parallel computing, memory layout, web security, and numerical methods.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

Build correct WebGPU device, resource, pipeline, rendering, and compute lifecycles.

Write validated WGSL kernels and reason about memory, workgroups, synchronization, and precision.

Operate browser-local visualization, simulation, data processing, and AI workloads responsibly.

Measure performance without creating GPU stalls, excessive copies, memory leaks, or UI starvation.

Handle security, privacy, device loss, feature diversity, accessibility, and fallback.

How to use this guide

Read the guide as a progressive system rather than a disconnected encyclopedia. Foundational material establishes the mental model; practitioner sections convert it into repeatable implementation and operations; advanced sections expose architecture, scale, security, and failure behavior; frontier sections describe technologies whose evidence or maturity is still evolving.

Suggested reading path
New practitioners should follow the chapters in order and complete the laboratories. Experienced engineers may begin with the architecture, failure, validation, or frontier sections, then use the related-document map to deepen a specific topic.

Learning and assurance model

Level	Reader outcome
L1 - Foundational	Terminology, first principles, component roles, packet or state flow, and simple working examples.
L2 - Practitioner	Implementation, configuration, automation, testing, troubleshooting, and operational evidence.
L3 - Advanced	Architecture, scale, concurrency, resilience, threat models, performance, and complex migrations.
L4 - Frontier	Emerging systems, research directions, technical uncertainty, readiness gates, and adoption signals.

Contents

CHAPTER 01

WebGPU architecture, security, and lifecycle

Treat GPU access as a powerful capability inside a hostile and diverse web environment.

Chapter operating position
Treat GPU access as a powerful capability inside a hostile and diverse web environment.

1.1 Adapter and device acquisition

Adapter and device acquisition must be engineered as an observable mechanism rather than a frontier label or speculative promise. Request power and feature preferences, inspect limits, handle unavailable capability, and avoid fingerprinting assumptions. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

- Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for adapter and device acquisition.
- Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.
- Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.
- Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.
- Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.
- Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Failure patterns

- Treating adapter and device acquisition as mature because a component or laboratory demonstration exists.
- Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.
- Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.
- Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.
- Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.
- Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for adapter and device acquisition.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

1.2 Queue and asynchronous lifecycle

Queue and asynchronous lifecycle must be engineered as an observable mechanism rather than a frontier label or speculative promise. Submit command buffers, await completion, handle asynchronous errors, validation, and device loss. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

- Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for queue and asynchronous lifecycle.
- Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.
- Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.
- Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.
- Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.
- Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Implementation sketch

```
webgpu_compute:
  feature_profile: baseline
  buffers:
    input: {usage: [storage, copy_dst], bytes: 4194304}
    output: {usage: [storage, copy_src], bytes: 1024}
  pipeline:
    shader: reduction.wgsl
    workgroup_size: 256
  validation:
    cpu_reference: required
    tolerance: 1e-5
  fallback: wasm-simd
  privacy:
    hardware-identifiers: not-collected
```

Failure patterns

Treating queue and asynchronous lifecycle as mature because a component or laboratory demonstration exists.

Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.

Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.

Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.

Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.

Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for queue and asynchronous lifecycle.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

1.3 Security and privacy model

Security and privacy model must be engineered as an observable mechanism rather than a frontier label or speculative promise. Respect secure contexts, cross-origin isolation, initialization, zeroing, bounds, timing, and hardware fingerprint concerns. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for security and privacy model.

Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.

Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.

Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.

Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.

Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Failure patterns

Treating security and privacy model as mature because a component or laboratory demonstration exists.

Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.

Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.

Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.

Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.

Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for security and privacy model.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

1.4 Progressive enhancement

Progressive enhancement must be engineered as an observable mechanism rather than a frontier label or speculative promise. Detect capability, select features, provide CPU or WebGL fallback, and preserve user control. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for progressive enhancement.

Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.

Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.

Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.

Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.

Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Failure patterns

Treating progressive enhancement as mature because a component or laboratory demonstration exists.

Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.

Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.

Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.

Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.

Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for progressive enhancement.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

Chapter 01 laboratory and review

Practical laboratory
Build a device lifecycle wrapper that handles unsupported features, validation error, device loss, and fallback.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore capability profiles for portable high-performance web applications.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 02

Buffers, textures, bindings, and memory

Represent data in GPU-compatible layouts and lifecycles.

Chapter operating position
Represent data in GPU-compatible layouts and lifecycles.

2.1 Buffer creation and usage

Buffer creation and usage must be engineered as an observable mechanism rather than a frontier label or speculative promise. Choose size, alignment, usage flags, mapping, staging, copies, storage, uniforms, vertices, indexes, and indirect arguments. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

- Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for buffer creation and usage.
- Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.
- Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.
- Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.
- Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.
- Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Failure patterns

- Treating buffer creation and usage as mature because a component or laboratory demonstration exists.
- Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.
- Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.
- Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.
- Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.
- Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for buffer creation and usage.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

2.2 Textures and views

Textures and views must be engineered as an observable mechanism rather than a frontier label or speculative promise. Manage dimensions, formats, mip levels, samples, storage textures, render attachments, copies, and presentation. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

- Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for textures and views.
- Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.
- Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.
- Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.
- Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.
- Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Implementation sketch

```
webgpu_compute:
  feature_profile: baseline
  buffers:
    input: {usage: [storage, copy_dst], bytes: 4194304}
    output: {usage: [storage, copy_src], bytes: 1024}
  pipeline:
    shader: reduction.wgsl
    workgroup_size: 256
  validation:
    cpu_reference: required
    tolerance: 1e-5
  fallback: wasm-simd
  privacy:
    hardware-identifiers: not-collected
```

Failure patterns

Treating textures and views as mature because a component or laboratory demonstration exists.

Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.

Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.

Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.

Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.

Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for textures and views.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

2.3 Bind groups and layouts

Bind groups and layouts must be engineered as an observable mechanism rather than a frontier label or speculative promise. Define resources, visibility, dynamic offsets, arrays, samplers, storage, and pipeline compatibility. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for bind groups and layouts.

Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.

Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.

Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.

Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.

Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Failure patterns

Treating bind groups and layouts as mature because a component or laboratory demonstration exists.

Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.

Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.

Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.

Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.

Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for bind groups and layouts.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

2.4 Memory ownership and lifetime

Memory ownership and lifetime must be engineered as an observable mechanism rather than a frontier label or speculative promise. Avoid use-before-ready, mapping conflicts, stale resources, excessive allocation, fragmentation, and leaks. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for memory ownership and lifetime.

Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.

Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.

Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.

Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.

Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Failure patterns

Treating memory ownership and lifetime as mature because a component or laboratory demonstration exists.

Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.

Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.

Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.

Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.

Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for memory ownership and lifetime.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

Chapter 02 laboratory and review

Practical laboratory
Implement a resource allocator and verify alignment, usage compatibility, copy bounds, and destruction.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore suballocation, sparse resources, external memory, and zero-copy browser pipelines.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 03

WGSL types, execution, and correctness

Write portable GPU programs whose behavior can be validated.

Chapter operating position
Write portable GPU programs whose behavior can be validated.

3.1 Types and address spaces

Types and address spaces must be engineered as an observable mechanism rather than a frontier label or speculative promise. Use scalars, vectors, matrices, arrays, structs, pointers, private, function, workgroup, uniform, storage, and handle spaces. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

- Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for types and address spaces.
- Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.
- Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.
- Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.
- Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.
- Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Failure patterns

- Treating types and address spaces as mature because a component or laboratory demonstration exists.
- Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.
- Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.
- Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.
- Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.
- Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for types and address spaces.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

3.2 Entry points and interfaces

Entry points and interfaces must be engineered as an observable mechanism rather than a frontier label or speculative promise. Define vertex, fragment, compute stages, built-ins, locations, bindings, interpolation, and pipeline linkage. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

- Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for entry points and interfaces.
- Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.
- Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.
- Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.
- Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.
- Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Implementation sketch

```
webgpu_compute:
  feature_profile: baseline
  buffers:
    input: {usage: [storage, copy_dst], bytes: 4194304}
    output: {usage: [storage, copy_src], bytes: 1024}
  pipeline:
    shader: reduction.wgsl
    workgroup_size: 256
  validation:
    cpu_reference: required
    tolerance: 1e-5
  fallback: wasm-simd
  privacy:
    hardware-identifiers: not-collected
```

Failure patterns

Treating entry points and interfaces as mature because a component or laboratory demonstration exists.

Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.

Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.

Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.

Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.

Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for entry points and interfaces.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

3.3 Control flow and numerical behavior

Control flow and numerical behavior must be engineered as an observable mechanism rather than a frontier label or speculative promise. Handle branches, loops, indexing, conversion, precision, overflow, derivatives, and undefined or implementation-dependent behavior. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for control flow and numerical behavior.

Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.

Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.

Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.

Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.

Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Failure patterns

Treating control flow and numerical behavior as mature because a component or laboratory demonstration exists.

Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.

Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.

Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.

Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.

Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for control flow and numerical behavior.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

3.4 Atomics, barriers, and races

Atomics, barriers, and races must be engineered as an observable mechanism rather than a frontier label or speculative promise. Coordinate workgroup memory, storage atomics, execution barriers, memory barriers, and data-race avoidance. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for atomics, barriers, and races.

Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.

Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.

Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.

Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.

Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Failure patterns

Treating atomics, barriers, and races as mature because a component or laboratory demonstration exists.

Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.

Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.

Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.

Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.

Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for atomics, barriers, and races.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

Chapter 03 laboratory and review

Practical laboratory
Write and test a WGSL reduction kernel with bounds checks, workgroup synchronization, and CPU reference validation.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore formally checked shader kernels and portable numerical contracts.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 04

Rendering pipelines and visualization

Create modern visual systems with explicit data and frame orchestration.

Chapter operating position
Create modern visual systems with explicit data and frame orchestration.

4.1 Vertex and index processing

Vertex and index processing must be engineered as an observable mechanism rather than a frontier label or speculative promise. Build geometry buffers, instance data, transforms, indirect draws, culling, and batching. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

- Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for vertex and index processing.
- Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.
- Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.
- Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.
- Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.
- Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Failure patterns

- Treating vertex and index processing as mature because a component or laboratory demonstration exists.
- Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.
- Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.
- Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.
- Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.
- Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for vertex and index processing.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

4.2 Rasterization and fragments

Rasterization and fragments must be engineered as an observable mechanism rather than a frontier label or speculative promise. Configure primitives, depth, stencil, multisampling, blending, attachments, and color spaces. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

- Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for rasterization and fragments.
- Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.
- Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.
- Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.
- Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.
- Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Implementation sketch

```
webgpu_compute:
  feature_profile: baseline
  buffers:
    input: {usage: [storage, copy_dst], bytes: 4194304}
    output: {usage: [storage, copy_src], bytes: 1024}
  pipeline:
    shader: reduction.wgsl
    workgroup_size: 256
  validation:
    cpu_reference: required
    tolerance: 1e-5
  fallback: wasm-simd
  privacy:
    hardware-identifiers: not-collected
```

Failure patterns

Treating rasterization and fragments as mature because a component or laboratory demonstration exists.

Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.

Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.

Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.

Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.

Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for rasterization and fragments.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

4.3 Materials and lighting

Materials and lighting must be engineered as an observable mechanism rather than a frontier label or speculative promise. Use textures, samplers, physically based shading, environment data, storage resources, and compute preprocessing. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for materials and lighting.

Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.

Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.

Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.

Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.

Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Failure patterns

Treating materials and lighting as mature because a component or laboratory demonstration exists.

Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.

Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.

Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.

Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.

Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for materials and lighting.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

4.4 Visualization and interaction

Visualization and interaction must be engineered as an observable mechanism rather than a frontier label or speculative promise. Render dense operational data, charts, graphs, spatial scenes, picking, overlays, and accessible alternatives. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for visualization and interaction.

Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.

Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.

Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.

Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.

Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Failure patterns

Treating visualization and interaction as mature because a component or laboratory demonstration exists.

Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.

Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.

Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.

Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.

Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for visualization and interaction.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

Chapter 04 laboratory and review

Practical laboratory
Build a dense technical visualization and measure frame time, upload cost, draw count, and interaction latency.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore browser-based digital twins and immersive operational environments.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 05

Compute pipelines and parallel algorithms

Use the browser GPU for non-rendering workloads safely.

Chapter operating position
Use the browser GPU for non-rendering workloads safely.

5.1 Dispatch and workgroups

Dispatch and workgroups must be engineered as an observable mechanism rather than a frontier label or speculative promise. Map problem dimensions to workgroups, invocations, limits, occupancy, local memory, and tails. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

- Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for dispatch and workgroups.
- Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.
- Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.
- Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.
- Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.
- Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Failure patterns

- Treating dispatch and workgroups as mature because a component or laboratory demonstration exists.
- Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.
- Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.
- Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.
- Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.
- Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for dispatch and workgroups.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

5.2 Parallel primitives

Parallel primitives must be engineered as an observable mechanism rather than a frontier label or speculative promise. Implement maps, reductions, scans, histograms, sorting, matrix operations, image processing, and simulation. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

- Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for parallel primitives.
- Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.
- Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.
- Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.
- Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.
- Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Implementation sketch

```
webgpu_compute:
  feature_profile: baseline
  buffers:
    input: {usage: [storage, copy_dst], bytes: 4194304}
    output: {usage: [storage, copy_src], bytes: 1024}
  pipeline:
    shader: reduction.wgsl
    workgroup_size: 256
  validation:
    cpu_reference: required
    tolerance: 1e-5
  fallback: wasm-simd
  privacy:
    hardware-identifiers: not-collected
```

Failure patterns

Treating parallel primitives as mature because a component or laboratory demonstration exists.

Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.

Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.

Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.

Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.

Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for parallel primitives.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

5.3 Data movement and staging

Data movement and staging must be engineered as an observable mechanism rather than a frontier label or speculative promise. Minimize CPU-GPU transfer, use persistent buffers, batching, readback, double buffering, and asynchronous flow. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for data movement and staging.

Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.

Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.

Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.

Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.

Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Failure patterns

Treating data movement and staging as mature because a component or laboratory demonstration exists.

Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.

Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.

Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.

Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.

Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for data movement and staging.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

5.4 Correctness and determinism

Correctness and determinism must be engineered as an observable mechanism rather than a frontier label or speculative promise. Compare CPU references, handle floating-point order, race conditions, boundaries, and reproducibility. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for correctness and determinism.

Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.

Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.

Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.

Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.

Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Failure patterns

Treating correctness and determinism as mature because a component or laboratory demonstration exists.

Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.

Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.

Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.

Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.

Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for correctness and determinism.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

Chapter 05 laboratory and review

Practical laboratory
Implement a compute pipeline and compare GPU versus CPU across small, crossover, and large workloads.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore general-purpose browser supercomputing and distributed WebGPU collaboration.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 06

Browser-local AI and WebNN interoperability

Choose the right browser accelerator interface for AI workloads.

Chapter operating position
Choose the right browser accelerator interface for AI workloads.

6.1 Tensor execution with WebGPU

Tensor execution with WebGPU must be engineered as an observable mechanism rather than a frontier label or speculative promise. Implement or host kernels for attention, convolution, normalization, activation, quantization, and sampling. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

- Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for tensor execution with webgpu.
- Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.
- Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.
- Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.
- Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.
- Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Failure patterns

- Treating tensor execution with webgpu as mature because a component or laboratory demonstration exists.
- Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.
- Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.
- Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.
- Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.
- Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for tensor execution with webgpu.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

6.2 WebNN graph execution

WebNN graph execution must be engineered as an observable mechanism rather than a frontier label or speculative promise. Use hardware-agnostic neural graph operators, contexts, tensors, device preferences, and implementation limits. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

- Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for webnn graph execution.
- Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.
- Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.
- Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.
- Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.
- Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Implementation sketch

```
webgpu_compute:
  feature_profile: baseline
  buffers:
    input: {usage: [storage, copy_dst], bytes: 4194304}
    output: {usage: [storage, copy_src], bytes: 1024}
  pipeline:
    shader: reduction.wgsl
    workgroup_size: 256
  validation:
    cpu_reference: required
    tolerance: 1e-5
  fallback: wasm-simd
  privacy:
    hardware-identifiers: not-collected
```

Failure patterns

Treating webnn graph execution as mature because a component or laboratory demonstration exists.

Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.

Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.

Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.

Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.

Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for webnn graph execution.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

6.3 WebGPU and WebNN composition

WebGPU and WebNN composition must be engineered as an observable mechanism rather than a frontier label or speculative promise. Share GPU devices or tensors where supported, split custom kernels from graph inference, and control copies. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for webgpu and webnn composition.

Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.

Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.

Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.

Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.

Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Failure patterns

Treating webgpu and webnn composition as mature because a component or laboratory demonstration exists.

Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.

Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.

Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.

Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.

Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for webgpu and webnn composition.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

6.4 Local model lifecycle

Local model lifecycle must be engineered as an observable mechanism rather than a frontier label or speculative promise. Load, cache, verify, quantize, stream, cancel, update, and delete models with storage and privacy policy. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for local model lifecycle.

Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.

Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.

Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.

Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.

Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Failure patterns

Treating local model lifecycle as mature because a component or laboratory demonstration exists.

Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.

Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.

Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.

Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.

Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for local model lifecycle.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

Chapter 06 laboratory and review

Practical laboratory
Design a browser-local inference pipeline that selects WebNN, WebGPU, or CPU based on operators, device, privacy, and performance.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore browser-local multimodal agents and private personal models.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 07

Performance engineering and diagnostics

Optimize the complete browser workload rather than isolated shader speed.

Chapter operating position
Optimize the complete browser workload rather than isolated shader speed.

7.1 CPU-GPU synchronization

CPU-GPU synchronization must be engineered as an observable mechanism rather than a frontier label or speculative promise. Avoid blocking readback, unnecessary queue waits, pipeline creation stalls, and frame serialization. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

- Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for cpu-gpu synchronization.
- Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.
- Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.
- Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.
- Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.
- Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Failure patterns

- Treating cpu-gpu synchronization as mature because a component or laboratory demonstration exists.
- Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.
- Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.
- Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.
- Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.
- Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for cpu-gpu synchronization.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

7.2 Pipeline and shader caching

Pipeline and shader caching must be engineered as an observable mechanism rather than a frontier label or speculative promise. Precompile, reuse layouts, control variants, warm critical paths, and manage browser cache behavior. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

- Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for pipeline and shader caching.
- Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.
- Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.
- Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.
- Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.
- Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Implementation sketch

```
webgpu_compute:
  feature_profile: baseline
  buffers:
    input: {usage: [storage, copy_dst], bytes: 4194304}
    output: {usage: [storage, copy_src], bytes: 1024}
  pipeline:
    shader: reduction.wgsl
    workgroup_size: 256
  validation:
    cpu_reference: required
    tolerance: 1e-5
  fallback: wasm-simd
  privacy:
    hardware-identifiers: not-collected
```

Failure patterns

Treating pipeline and shader caching as mature because a component or laboratory demonstration exists.

Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.

Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.

Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.

Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.

Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for pipeline and shader caching.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

7.3 Profiling and telemetry

Profiling and telemetry must be engineered as an observable mechanism rather than a frontier label or speculative promise. Measure frame, pass, dispatch, queue, upload, readback, memory, compilation, device loss, and user-perceived latency. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for profiling and telemetry.

Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.

Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.

Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.

Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.

Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Failure patterns

Treating profiling and telemetry as mature because a component or laboratory demonstration exists.

Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.

Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.

Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.

Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.

Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for profiling and telemetry.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

7.4 Power and thermal behavior

Power and thermal behavior must be engineered as an observable mechanism rather than a frontier label or speculative promise. Respect mobile battery, sustained load, throttling, background tabs, user settings, and graceful degradation. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for power and thermal behavior.

Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.

Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.

Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.

Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.

Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Failure patterns

Treating power and thermal behavior as mature because a component or laboratory demonstration exists.

Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.

Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.

Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.

Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.

Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for power and thermal behavior.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

Chapter 07 laboratory and review

Practical laboratory
Profile a WebGPU application and eliminate one upload bottleneck, one synchronization stall, and one memory leak.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore adaptive quality systems driven by energy, thermal, and interaction constraints.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 08

Deployment, accessibility, and frontier systems

Ship high-performance web applications without excluding users or hiding risk.

Chapter operating position
Ship high-performance web applications without excluding users or hiding risk.

8.1 Compatibility and testing

Compatibility and testing must be engineered as an observable mechanism rather than a frontier label or speculative promise. Use conformance behavior, browser matrices, operating systems, drivers, limits, fallbacks, and automated visual and numerical tests. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

- Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for compatibility and testing.
- Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.
- Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.
- Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.
- Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.
- Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Failure patterns

- Treating compatibility and testing as mature because a component or laboratory demonstration exists.
- Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.
- Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.
- Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.
- Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.
- Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for compatibility and testing.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

8.2 Accessibility and operational UX

Accessibility and operational UX must be engineered as an observable mechanism rather than a frontier label or speculative promise. Provide keyboard, screen-reader, reduced motion, contrast, textual data, focus, and nonvisual controls. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

- Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for accessibility and operational ux.
- Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.
- Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.
- Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.
- Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.
- Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Implementation sketch

```
webgpu_compute:
  feature_profile: baseline
  buffers:
    input: {usage: [storage, copy_dst], bytes: 4194304}
    output: {usage: [storage, copy_src], bytes: 1024}
  pipeline:
    shader: reduction.wgsl
    workgroup_size: 256
  validation:
    cpu_reference: required
    tolerance: 1e-5
  fallback: wasm-simd
  privacy:
    hardware-identifiers: not-collected
```

Failure patterns

Treating accessibility and operational ux as mature because a component or laboratory demonstration exists.

Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.

Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.

Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.

Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.

Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for accessibility and operational ux.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

8.3 Security and content supply chain

Security and content supply chain must be engineered as an observable mechanism rather than a frontier label or speculative promise. Validate shaders, models, assets, workers, origins, permissions, dependencies, and update integrity. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for security and content supply chain.

Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.

Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.

Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.

Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.

Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Failure patterns

Treating security and content supply chain as mature because a component or laboratory demonstration exists.

Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.

Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.

Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.

Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.

Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for security and content supply chain.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

8.4 Frontier applications

Frontier applications must be engineered as an observable mechanism rather than a frontier label or speculative promise. Evaluate browser-local CAD, scientific computing, simulation, media, AI, spatial interfaces, and sovereign web applications. The design should identify physical and logical assumptions, authoritative state, interfaces, identities, resource limits, uncertainty, telemetry, failure behavior, recovery, and acceptance evidence.

This guide traces the subject through the application data, browser policy, adapter and device, GPU resource, pipeline or shader, queue execution, result, diagnostics, and user experience chain. A laboratory demonstration, simulation result, standards reference, valid packet, successful kernel, or fluent AI interaction is not sufficient proof. Intended behavior must be reconciled with measured state, negative tests, degraded conditions, lifecycle constraints, complete-system cost, and user or mission outcomes.

Implementation begins with a small known-good baseline and explicit invariants. Introduce one device, algorithm, optical engine, robot skill, transfer path, component interface, GPU kernel, approval mode, or network feature at a time; preserve before-state evidence; test prohibited and failed behavior; and retain a reversible fallback.

Troubleshooting locates the first divergence from the expected state machine or physical model. Account for calibration, loss, noise, timing, version skew, hidden dependencies, resource exhaustion, stale state, synchronization, permissions, model uncertainty, supply-chain changes, and missing evidence. Closure requires causal explanation, reproducibility, validated recovery, and disclosed limitations.

Engineering practices

Define the objective, authority, physical assumptions, interfaces, resources, risks, and acceptance evidence for frontier applications.

Separate stable standards and deployable practice from drafts, experiments, prototypes, and research hypotheses.

Version hardware, firmware, software, models, schemas, calibration, data, policy, and evaluation artifacts.

Test nominal, prohibited, noisy, degraded, overloaded, disconnected, failed, recovery, and rollback conditions.

Preserve source, configuration, measurement, trace, artifact, approval, experiment, and user-outcome evidence.

Adopt through staged pilots, independent baselines, explicit exit criteria, and reversible architecture.

Failure patterns

Treating frontier applications as mature because a component or laboratory demonstration exists.

Comparing a specialized accelerator or protocol against a weak or incomplete classical baseline.

Ignoring physical loss, calibration, timing, packaging, power, thermal, network, and operational constraints.

Allowing AI, autonomy, local compute, or disconnected operation to bypass identity, policy, safety, and evidence controls.

Using average performance while hiding tails, failed trials, environmental sensitivity, or complete-system overhead.

Declaring adoption success without interoperability, maintenance, recovery, workforce, supplier, and exit evidence.

Validation and evidence

Method	Expected evidence
Examine	Architecture, standards maturity, physical assumptions, identity, interfaces, versions, resources, and risks for frontier applications.
Observe	Signals, states, measurements, packets, events, traces, resource use, errors, artifacts, approvals, and user-visible outcomes.
Test	Expected, prohibited, noisy, adversarial, overloaded, disconnected, partially failed, recovery, and rollback cases.
Measure	Correctness, fidelity, accuracy, latency, throughput, energy, availability, safety, cost, scalability, and operator effort.
Reconcile	Claimed capability against measured physical, software, network, security, operational, and user evidence.

Chapter 08 laboratory and review

Practical laboratory

Create a release qualification plan covering browser matrix, numerical correctness, accessibility, privacy, and performance.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore offline-first browser workstations combining WebGPU, WebNN, WASM, local data, and verifiable updates.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

Integrated learning roadmap

The guide is intended to be revisited. First establish fluency, then build a small implementation, operate it under failure, automate repeatable work, and finally evaluate emerging techniques against a stable baseline. Progress is demonstrated through evidence rather than reading completion alone.

Stage	Demonstrated capability
Foundation	Explain the system from first principles and trace its critical path without relying on product terminology.
Build	Implement the smallest representative system with explicit contracts, identities, telemetry, and tests.
Operate	Diagnose injected failures, recover safely, and preserve evidence of the causal chain.
Automate	Convert a proven manual workflow into bounded, idempotent, observable automation.
Architect	Design for scale, resilience, security, interoperability, and lifecycle change.
Explore	Evaluate frontier techniques using stated hypotheses, limitations, and adoption gates.

Source authority register

Source policy
Official standards, specifications, and project documentation remain with their issuing organizations. Mythos Systems provides original engineering interpretation, synthesis, implementation guidance, and relationship mapping. Source verification dates do not guarantee that an authority has not changed since review.

01. W3C - WebGPU
Role: Current browser GPU API, security, privacy, device, resource, pipeline, rendering, and compute semantics.
Verified: 2026-07-26
Official source: https://www.w3.org/TR/webgpu/

02. W3C - WebGPU Shading Language
Role: Current shading language, types, memory model, validation, execution, and security considerations.
Verified: 2026-07-26
Official source: https://www.w3.org/TR/WGSL/

03. W3C - Web Neural Network API
Role: Candidate Recommendation for browser-local neural-network acceleration and WebGPU interoperability.
Verified: 2026-07-26
Official source: https://www.w3.org/TR/webnn/

Authority application and refresh triggers

Authority class	Required library action
Protocols and specifications	Recheck interoperability, security considerations, defaults, and migration guidance when a referenced specification changes.
Security and assurance frameworks	Update control mappings, evidence expectations, risk language, and assessment procedures after material framework revision.
Languages, runtimes, and projects	Re-run examples, tests, deprecation checks, and operational recommendations against supported releases.
Benchmarks and evaluations	Re-execute claims when workloads, methods, models, compilers, drivers, hardware, or scoring rules change.
Operational evidence	Incorporate validated incidents, failure tests, reader corrections, and measured implementation outcomes into the next revision.

Limitations, freshness, and revision control

Boundary	Statement
Publication limitation	This guide synthesizes broad engineering practice. It does not replace product documentation, legal advice, safety certification, or environment-specific design review.
Evidence limitation	Not every pattern is benchmarked in every environment. Examples illustrate engineering methods and must be validated against actual versions, workloads, hardware, policy, and failure conditions.
Freshness limitation	Vendor behavior, software libraries, AI models, security threats, and emerging standards can change quickly. Volatile claims require source revalidation before operational adoption.
Adoption limitation	Frontier content is not an implicit adoption recommendation. Adoption requires defined value, qualification gates, controlled proof, rollback, ownership, and residual-risk acceptance.
Status boundary	Candidate Focused Field Guide describes the publication, not certification of a product, implementation, engineer, or organization.

Revision record

Version	Revision statement
1.0.0-candidate	Initial candidate living-guide edition. Published 2026-07-26; scheduled review 2026-10-26.

Search and relationship metadata

Dimension	Engineering position
Primary domain	MYTHOS-EMT - Emerging Technology & Sovereign Infrastructure
Secondary domain	MYTHOS-UX; MYTHOS-SWE; MYTHOS-AI
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	WebGPU, WGSL, WebNN, browser local compute, GPU compute, high performance web, browser AI, visualization, privacy, progressive enhancement
Canonical route	/library/field-guides/mythos-fg-frontier-0008/

Living publication

Corrections, expanded laboratories, improved diagrams, authority changes, and new engineering evidence should revise this permanent publication rather than create disconnected replacements.