CANONICAL LIVING OVERVIEW GUIDE

Neuromorphic Computing, Spiking Systems, and NPUs

A guide to event-driven neural computation, spiking models, neuromorphic hardware, Loihi-class systems, Lava, learning rules, sensors, NPUs, workload mapping, benchmarking, and adoption limitations.

PERMANENT ID
MYTHOS-FG-FRONTIER-0004

PUBLICATION CLASS
FIELD GUIDE

VERSION / STATUS
1.0.0-candidate / CANDIDATE

PERMANENT PUBLICATION IDENTITY

Neuromorphic Computing, Spiking Systems, and NPUs

Document ID
MYTHOS-FG-FRONTIER-0004

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-AI; MYTHOS-DAT; MYTHOS-ROB
Audience	AI, edge, embedded, robotics, accelerator, neuroscience-inspired computing, and research engineers.
Prerequisites	Neural networks, probability, differential equations, digital hardware, embedded systems, and Python.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

Explain event-driven neural computation and spiking-state dynamics.

Implement and evaluate leaky integrate-and-fire networks and event streams.

Compare neuromorphic chips, conventional NPUs, GPUs, CPUs, and mixed systems.

Map sensing, control, optimization, and sparse workloads to appropriate architectures.

Benchmark energy, latency, accuracy, adaptability, and total integration cost honestly.

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

How to use this guide 3

Contents 3

Chapter 01 - Neuromorphic principles and event-driven computation 5

Chapter 01 laboratory and review 10

Chapter 02 - Spiking models and learning rules 11

Chapter 02 laboratory and review 16

Chapter 03 - Neuromorphic hardware architecture 17

Chapter 03 laboratory and review 22

Chapter 04 - NPUs and heterogeneous edge acceleration 23

Chapter 04 laboratory and review 28

Chapter 05 - Software stacks, Lava, and application composition 29

Chapter 05 laboratory and review 34

Chapter 06 - Workloads and system patterns 35

Chapter 06 laboratory and review 40

Chapter 07 - Benchmarking and evaluation 41

Chapter 07 laboratory and review 46

Chapter 08 - Adoption, limitations, and frontier roadmap 47

Chapter 08 laboratory and review 52

Integrated learning roadmap 53

Source authority register 54

Limitations, freshness, and revision control 55

CHAPTER 01

Neuromorphic principles and event-driven computation

Define the computational model before discussing hardware claims.

Chapter operating position
Define the computational model before discussing hardware claims.

1.1 Sparse asynchronous events

Sparse asynchronous events must be engineered as an observable mechanism rather than a frontier label or speculative promise. Represent information through event timing, rate, population, address events, and local updates. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 sparse asynchronous events.
- 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 sparse asynchronous events 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 sparse asynchronous events.
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 Stateful neuron dynamics

Stateful neuron dynamics must be engineered as an observable mechanism rather than a frontier label or speculative promise. Model membrane potential, leakage, thresholds, reset, refractory behavior, and temporal integration. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 stateful neuron dynamics.
- 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

```
spiking_workload:
  encoding: temporal
  neuron: leaky-integrate-and-fire
  timestep_ms: 1
  threshold: 1.0
  decay: 0.92
  learning: reward-modulated-local
  baseline: quantized-npu
  metrics:
    - event-to-action-latency
    - task-accuracy
    - energy-per-decision
    - adaptation-time
```

Failure patterns

- Treating stateful neuron dynamics 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 stateful neuron dynamics.
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 Synapses and connectivity

Synapses and connectivity must be engineered as an observable mechanism rather than a frontier label or speculative promise. Use weighted events, delays, plasticity, fan-in, fan-out, sparse graphs, and locality. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 synapses and connectivity.

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 synapses and connectivity 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 synapses and connectivity.
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 Temporal and dynamical computation

Temporal and dynamical computation must be engineered as an observable mechanism rather than a frontier label or speculative promise. Treat time, oscillation, attractors, recurrence, and continuous interaction as first-class. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 temporal and dynamical computation.

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 temporal and dynamical computation 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 temporal and dynamical computation.
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
Implement a leaky integrate-and-fire neuron and network driven by sparse events, then measure activity and 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 continuous-time neural fields and hybrid analog-digital neuromorphic systems.

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

Spiking models and learning rules

Choose models that match the required fidelity and hardware cost.

Chapter operating position
Choose models that match the required fidelity and hardware cost.

2.1 Neuron model families

Neuron model families must be engineered as an observable mechanism rather than a frontier label or speculative promise. Compare integrate-and-fire, LIF, adaptive, Izhikevich, compartmental, stochastic, and surrogate models. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 neuron model families.
- 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 neuron model families 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 neuron model families.
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 Spike coding

Spike coding must be engineered as an observable mechanism rather than a frontier label or speculative promise. Evaluate rate, latency, rank-order, temporal, population, phase, and event-based representations. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 spike coding.
- 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

```
spiking_workload:
  encoding: temporal
  neuron: leaky-integrate-and-fire
  timestep_ms: 1
  threshold: 1.0
  decay: 0.92
  learning: reward-modulated-local
  baseline: quantized-npu
  metrics:
    - event-to-action-latency
    - task-accuracy
    - energy-per-decision
    - adaptation-time
```

Failure patterns

- Treating spike coding 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 spike coding.
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 Local plasticity

Local plasticity must be engineered as an observable mechanism rather than a frontier label or speculative promise. Use Hebbian, STDP, reward-modulated, homeostatic, and eligibility-trace mechanisms. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 plasticity.

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 plasticity 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 plasticity.
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 Training and conversion

Training and conversion must be engineered as an observable mechanism rather than a frontier label or speculative promise. Compare surrogate gradients, backpropagation through time, ANN-to-SNN conversion, local learning, and reservoir methods. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 training and conversion.

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 training and conversion 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 training and conversion.
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
Train or tune a small spiking classifier and compare rate and temporal coding under noise.

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 credit assignment through local learning and dendritic computation.

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

Neuromorphic hardware architecture

Understand how event-driven computation is realized physically.

Chapter operating position
Understand how event-driven computation is realized physically.

3.1 Digital neuromorphic cores

Digital neuromorphic cores must be engineered as an observable mechanism rather than a frontier label or speculative promise. Examine neuron and synapse engines, event routers, local memory, programmability, timing, and scaling. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 digital neuromorphic cores.
- 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 digital neuromorphic cores 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 digital neuromorphic cores.
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 Analog and mixed-signal systems

Analog and mixed-signal systems must be engineered as an observable mechanism rather than a frontier label or speculative promise. Evaluate subthreshold dynamics, mismatch, noise, calibration, density, and energy. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 analog and mixed-signal systems.
- 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

```
spiking_workload:
  encoding: temporal
  neuron: leaky-integrate-and-fire
  timestep_ms: 1
  threshold: 1.0
  decay: 0.92
  learning: reward-modulated-local
  baseline: quantized-npu
  metrics:
    - event-to-action-latency
    - task-accuracy
    - energy-per-decision
    - adaptation-time
```

Failure patterns

- Treating analog and mixed-signal systems 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 analog and mixed-signal systems.
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 Loihi-class architecture

Loihi-class architecture must be engineered as an observable mechanism rather than a frontier label or speculative promise. Study programmable compartments, learning engines, asynchronous mesh, event routing, and board-scale systems. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 loihi-class architecture.

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 loihi-class architecture 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 loihi-class architecture.
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 Sensors and closed-loop hardware

Sensors and closed-loop hardware must be engineered as an observable mechanism rather than a frontier label or speculative promise. Integrate event cameras, cochleas, tactile sensors, motors, and realtime 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 sensors and closed-loop hardware.

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 sensors and closed-loop hardware 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 sensors and closed-loop hardware.
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
Map a spiking network onto a multi-core neuromorphic topology and identify communication and state bottlenecks.

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 memristive synapses, 3D integration, and chip-scale sensor-compute systems.

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

NPUs and heterogeneous edge acceleration

Differentiate mainstream neural accelerators from neuromorphic systems.

Chapter operating position
Differentiate mainstream neural accelerators from neuromorphic systems.

4.1 NPU dataflow

NPU dataflow must be engineered as an observable mechanism rather than a frontier label or speculative promise. Understand tensors, MAC arrays, systolic execution, SRAM, quantization, operator support, and compiler graphs. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 npu dataflow.
- 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 npu dataflow 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 npu dataflow.
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 Neuromorphic versus NPU

Neuromorphic versus NPU must be engineered as an observable mechanism rather than a frontier label or speculative promise. Compare event sparsity, temporal state, programmability, accuracy, throughput, batching, and ecosystem. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 neuromorphic versus npu.
- 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

```
spiking_workload:
  encoding: temporal
  neuron: leaky-integrate-and-fire
  timestep_ms: 1
  threshold: 1.0
  decay: 0.92
  learning: reward-modulated-local
  baseline: quantized-npu
  metrics:
    - event-to-action-latency
    - task-accuracy
    - energy-per-decision
    - adaptation-time
```

Failure patterns

- Treating neuromorphic versus npu 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 neuromorphic versus npu.
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 CPU, GPU, NPU, and neuromorphic partitioning

CPU, GPU, NPU, and neuromorphic partitioning must be engineered as an observable mechanism rather than a frontier label or speculative promise. Assign preprocessing, dense inference, sparse events, control, learning, and orchestration. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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, npu, and neuromorphic partitioning.

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, npu, and neuromorphic partitioning 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, npu, and neuromorphic partitioning.
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 On-device constraints

On-device constraints must be engineered as an observable mechanism rather than a frontier label or speculative promise. Manage memory, power, thermal, latency, privacy, model updates, sensor bandwidth, and reliability. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 on-device constraints.

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 on-device constraints 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 on-device constraints.
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
Design a heterogeneous edge pipeline and justify which stages run on CPU, GPU/NPU, and neuromorphic hardware.

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 runtimes that move work between dense and event-driven accelerators.

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

Software stacks, Lava, and application composition

Build portable software despite hardware diversity.

Chapter operating position
Build portable software despite hardware diversity.

5.1 Process and port abstractions

Process and port abstractions must be engineered as an observable mechanism rather than a frontier label or speculative promise. Represent computation as interacting stateful processes, messages, models, and execution protocols. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 process and port abstractions.
- 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 process and port abstractions 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 process and port abstractions.
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 Lava execution models

Lava execution models must be engineered as an observable mechanism rather than a frontier label or speculative promise. Separate application, process model, compiler, runtime, simulator, CPU backend, and neuromorphic backend. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 lava execution models.
- 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

```
spiking_workload:
  encoding: temporal
  neuron: leaky-integrate-and-fire
  timestep_ms: 1
  threshold: 1.0
  decay: 0.92
  learning: reward-modulated-local
  baseline: quantized-npu
  metrics:
    - event-to-action-latency
    - task-accuracy
    - energy-per-decision
    - adaptation-time
```

Failure patterns

- Treating lava execution models 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 lava execution models.
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 Graph partition and mapping

Graph partition and mapping must be engineered as an observable mechanism rather than a frontier label or speculative promise. Allocate neurons, synapses, state, learning, routes, and resources across cores and chips. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 graph partition and mapping.

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 graph partition and mapping 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 graph partition and mapping.
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 Debugging and observability

Debugging and observability must be engineered as an observable mechanism rather than a frontier label or speculative promise. Inspect spikes, states, weights, events, congestion, timing, energy, and divergence. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 debugging and observability.

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 debugging and observability 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 debugging and observability.
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
Build a small Lava-style process graph and test equivalent behavior across a Python simulation and constrained backend model.

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 open intermediate representations for neuro-inspired computing.

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

Workloads and system patterns

Use neuromorphic computing where event-driven dynamics provide measurable value.

Chapter operating position
Use neuromorphic computing where event-driven dynamics provide measurable value.

6.1 Event-based perception

Event-based perception must be engineered as an observable mechanism rather than a frontier label or speculative promise. Process dynamic vision, audio, tactile, radar, and sparse sensor streams with low 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 event-based perception.
- 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 event-based perception 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 event-based perception.
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 Control and robotics

Control and robotics must be engineered as an observable mechanism rather than a frontier label or speculative promise. Implement reflexes, localization support, adaptive control, navigation, and closed-loop learning. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 and robotics.
- 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

```
spiking_workload:
  encoding: temporal
  neuron: leaky-integrate-and-fire
  timestep_ms: 1
  threshold: 1.0
  decay: 0.92
  learning: reward-modulated-local
  baseline: quantized-npu
  metrics:
    - event-to-action-latency
    - task-accuracy
    - energy-per-decision
    - adaptation-time
```

Failure patterns

- Treating control and robotics 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 and robotics.
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 Optimization and search

Optimization and search must be engineered as an observable mechanism rather than a frontier label or speculative promise. Map graph, constraint, scheduling, routing, and combinatorial problems to dynamical systems. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 optimization and search.

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 optimization and search 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 optimization and search.
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 Adaptive edge intelligence

Adaptive edge intelligence must be engineered as an observable mechanism rather than a frontier label or speculative promise. Use continual adaptation, anomaly detection, personalization, and low-power always-on 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 adaptive edge intelligence.

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 adaptive edge intelligence 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 adaptive edge intelligence.
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
Benchmark an event-based workload against a frame-based or dense baseline using identical outcome criteria.

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 embodied world models built from asynchronous multimodal event streams.

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

Benchmarking and evaluation

Measure complete-system value rather than isolated chip efficiency.

Chapter operating position
Measure complete-system value rather than isolated chip efficiency.

7.1 Accuracy and task quality

Accuracy and task quality must be engineered as an observable mechanism rather than a frontier label or speculative promise. Use representative data, temporal metrics, calibration, robustness, adaptation, and failure slices. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 accuracy and task quality.
- 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 accuracy and task quality 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 accuracy and task quality.
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 Latency and throughput

Latency and throughput must be engineered as an observable mechanism rather than a frontier label or speculative promise. Measure event-to-action time, jitter, sustained rate, queueing, and overload 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 latency and throughput.
- 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

```
spiking_workload:
  encoding: temporal
  neuron: leaky-integrate-and-fire
  timestep_ms: 1
  threshold: 1.0
  decay: 0.92
  learning: reward-modulated-local
  baseline: quantized-npu
  metrics:
    - event-to-action-latency
    - task-accuracy
    - energy-per-decision
    - adaptation-time
```

Failure patterns

- Treating latency and throughput 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 latency and throughput.
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 Energy and resource accounting

Energy and resource accounting must be engineered as an observable mechanism rather than a frontier label or speculative promise. Include sensors, host CPU, conversion, communication, memory, cooling, and idle power. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 energy and resource accounting.

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 energy and resource accounting 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 energy and resource accounting.
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 Reproducibility and comparison

Reproducibility and comparison must be engineered as an observable mechanism rather than a frontier label or speculative promise. Record hardware, software, mapping, seeds, time scale, precision, data, and classical baseline. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 reproducibility and comparison.

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 reproducibility and comparison 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 reproducibility and comparison.
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
Create a benchmark report comparing neuromorphic, NPU, and CPU implementations of one edge task.

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

Develop workload-neutral neuromorphic benchmarks and energy-normalized adaptation metrics.

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

Adoption, limitations, and frontier roadmap

Build a disciplined path through an immature and heterogeneous ecosystem.

Chapter operating position
Build a disciplined path through an immature and heterogeneous ecosystem.

8.1 Capability and maturity

Capability and maturity must be engineered as an observable mechanism rather than a frontier label or speculative promise. Separate research demonstrations, development boards, supported workloads, scalable systems, and products. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 capability and maturity.
- 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 capability and maturity 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 capability and maturity.
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 Ecosystem and portability

Ecosystem and portability must be engineered as an observable mechanism rather than a frontier label or speculative promise. Assess tooling, debuggability, model conversion, skills, hardware access, standards, and vendor dependency. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 ecosystem and portability.
- 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

```
spiking_workload:
  encoding: temporal
  neuron: leaky-integrate-and-fire
  timestep_ms: 1
  threshold: 1.0
  decay: 0.92
  learning: reward-modulated-local
  baseline: quantized-npu
  metrics:
    - event-to-action-latency
    - task-accuracy
    - energy-per-decision
    - adaptation-time
```

Failure patterns

- Treating ecosystem and portability 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 ecosystem and portability.
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 Safety and assurance

Safety and assurance must be engineered as an observable mechanism rather than a frontier label or speculative promise. Validate temporal behavior, adaptation bounds, sensor attacks, state reset, updates, and fail-safe 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 safety and assurance.

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 safety and assurance 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 safety and assurance.
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 Adoption decision

Adoption decision must be engineered as an observable mechanism rather than a frontier label or speculative promise. Compare performance, energy, lifecycle, integration, opportunity cost, and fallback architecture. 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 sensor or input event, encoding, neuron and synapse state, event routing, learning, output action, measurement, and workload outcome 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 adoption decision.
- 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 adoption decision 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 adoption decision.
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
Write an adoption decision for a robotics or sensing workload with explicit exit criteria.

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 sovereign neuromorphic appliances and self-organizing edge intelligence with bounded learning.

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. Intel Labs - Neuromorphic Computing and Engineering with AI
Role: Loihi 2, neuromorphic systems, application research, and Lava integration.
Verified: 2026-07-26
Official source: https://www.intel.com/content/www/us/en/research/neuromorphic-computing.html

02. Intel - Hala Point Neuromorphic System
Role: Large-scale Loihi 2 system and neuromorphic workload context.
Verified: 2026-07-26
Official source: https://newsroom.intel.com/artificial-intelligence/intel-builds-worlds-largest-neuromorphic-system-to-enable-more-sustainable-ai

03. Lava Project - Lava Software Framework
Role: Open software framework for neuro-inspired applications and heterogeneous neuromorphic hardware.
Verified: 2026-07-26
Official source: https://lava-nc.org/

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-AI; MYTHOS-DAT; MYTHOS-ROB
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	neuromorphic computing, spiking neural networks, Loihi 2, Lava, NPU, event-driven AI, edge intelligence, event camera, low power, robotics
Canonical route	/library/field-guides/mythos-fg-frontier-0004/

Living publication

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