

OPEN FOUNDATION GUIDE · 005

# HEDL: the complete foundation guide

Mechanism, artefact, interfaces, stack joins, failure, reproducibility, performance method, security, deployment, adoption, and exit for HEDL.

HEDL

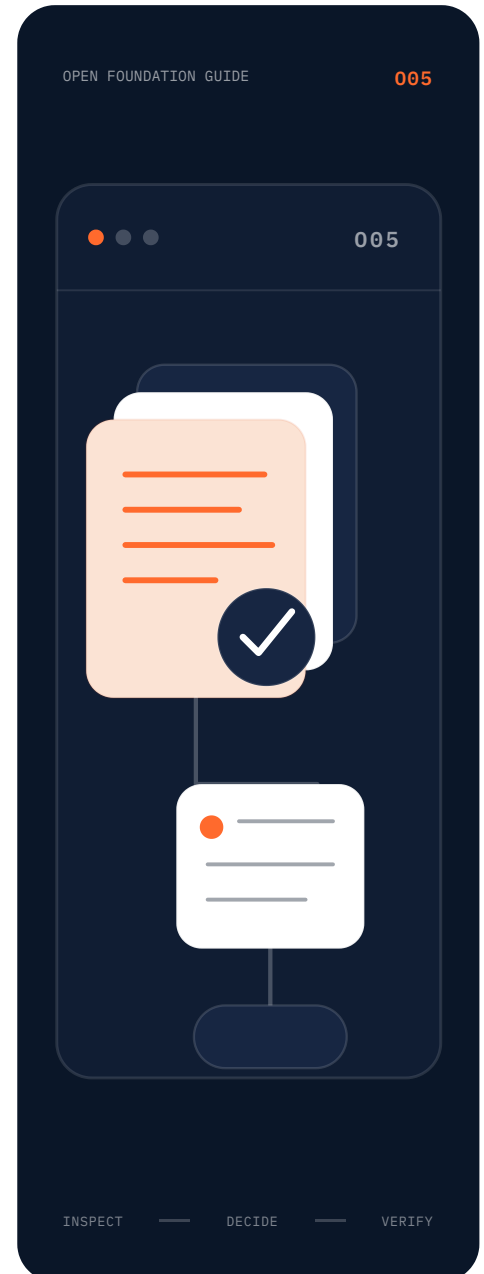
MECHANISM

ARTEFACT

REPRODUCIBILITY

SECURITY

EXIT



Read this when the name HEDL is not enough and the team needs to inspect the component contract and reproduce its claim.

005

## CONTENTS

# HEDL, from mechanism to independent check

The guide uses one illustrative exercise: an agent/tool workflow sends repetitive structured payloads through a token-billed model path; the supported data round-trips through local conversion and the task remains equivalent.

## PRIMARY READERS

Engineers, maintainers and independent reviewers

## READING DEPTH

Deep technical

## DECISION THIS SUPPORTS

A component-level guide for teams deciding whether and how to rely on HEDL for compact, local, lossless exchange for model and agent data paths.

**01 Read this first** 03–04  
Scope, reader decision, vocabulary, source and claim status

**02 Mechanism and artefact** 05–12  
Responsibility, input, pipeline, manifest, interfaces, stack joins, and failure

**03 Evaluation and adoption** 13–19  
Reproducibility, performance, security, deployment, integration, diligence, and exit

**04 Sources and next action** 22  
verification records and component proof package

**05 Subject index** 20–21  
An alphabetical finder for concepts, controls, products, and decisions.

## READING RULE

The published Hedl open-foundation page owns factual statements. Illustrative envelopes and evaluation procedures show what a credible integration should specify; they are not invented project APIs.

## SOURCE DISCIPLINE

Dweve-specific facts are taken directly from the English website text audited 2026-07-20; web-navigation wording is humanised for print. Evaluation guidance is editorial, and scenarios and derived visuals are illustrative. Unsupported synthesis is replaced by canonical copy or omitted.

READ THIS FIRST • DECISION

# Who should read the HEDL guide

Engineers, architects, security and assurance teams, product owners, open-source reviewers, and operators can use this guide to evaluate compact, local, lossless exchange for model and agent data paths.

Every AI call is priced by the token, and much of that budget is syntax rather than data. HEDL sent the same request in 56 percent fewer tokens in the measured run and showed a +10.3pp accuracy difference from JSON there. That is evidence from this setup, alongside a lower bill, without changing downstream systems.

The worked example begins when an agent/tool workflow sends repetitive structured payloads through a token-billed model path. The expected normal behaviour is: the supported data round-trips through local conversion and the task remains equivalent. The fault case is: an unsupported value/schema or framing edge enters. A sound result is: conversion refuses or exposes the incompatibility instead of silently changing the payload. These statements are illustrative evaluation framing, not a customer outcome.

Verify current source/repository and release status, licence, documentation, supported interfaces/platforms, product integration, benchmark method, security, deployment artefacts, support, and rights at the decision date. The website is the sole factual source for this edition; the installed component supplies final proof.

## FOUNDATION

### HEDL

compact, local, lossless exchange for model and agent data paths

## OBJECT

### compact lossless exchange payload

The bounded technical artefact.

## FAULT

### Deliberate negative case

an unsupported value/schema or framing edge enters

## DECISION

### Fit + reproduce

Integrate, operate, verify and exit.

## PRIMARY VERIFICATION

Round-trip the production payload corpus byte/semantic-equivalently under the documented contract; compare schema edge cases; run direct and proxy paths; reproduce token and accuracy measurements with the named models/tasks; remove the proxy; and confirm the application returns to JSON.

[READ THIS FIRST](#) • [TERMS](#)

# Vocabulary and claim boundary

Foundation, project, surface, artefact, manifest, verifier, benchmark, product integration, and commercial/source rights answer different questions.

## Mechanism and artefact

HEDL adds one compact exchange layer before the model while every database, report, and application continues speaking JSON. Run conversion locally, measure it on one workflow, and remove it without a data migration if the numbers do not hold.

## Project, product, and surface

The open-foundation description describes the project. A product may use it without exposing the same interface or licence. A library, CLI, service, sidecar, FFI, MCP or WASM build is a surface; surface availability and parity require current verification.

## Measured and assurance claims

A number remains tied to its corpus/input, hardware, version, configuration, method, baseline and limitation. Integrity, determinism, provenance, containment or offline checking support bounded assurance; they do not prove truth, correctness, legitimacy, legal compliance, fitness or commercial entitlement.

| TERM               | ANSWERS                       | DOES NOT ANSWER                     |
|--------------------|-------------------------------|-------------------------------------|
| <b>Foundation</b>  | Which mechanism?              | Who owns product work               |
| <b>Artefact</b>    | What can be consumed/checked? | Why purpose is legitimate           |
| <b>Surface</b>     | How invoked?                  | Semantic parity automatically       |
| <b>Open/source</b> | What can be inspected?        | Current release/right automatically |
| <b>Benchmark</b>   | What was measured?            | Production SLA/ROI                  |
| <b>Verifier</b>    | Which proposition holds?      | Every upstream premise is true      |

Keep these distinctions attached to every adoption claim.

### FOUNDATION COUNT WARNING

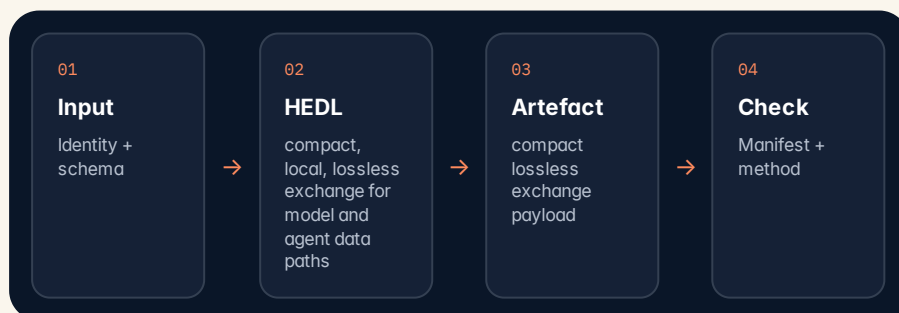
Only relationships asserted by the current product and open-source pages are drawn here. Absence from this ledger is not a claim that a foundation is unused.

01

MECHANISM AND ARTEFACT

# HEDL must earn its place at one technical join

Define the problem, input, mechanism, output artefact, interfaces, dependencies, failure, and evidence before discussing adoption.



The component contract used throughout the chapter.

## MECHANISM • RESPONSIBILITY

# HEDL solves one inspectable technical problem

HEDL is the foundation for compact, local, lossless exchange for model and agent data paths. Its value is the contract and artefact it contributes, not “open source” as a decorative label.

HEDL removes syntactic overhead while preserving the supported data semantics and restoring the original exchange shape. Conversion is local and bidirectional; a proxy can speak HEDL upstream and JSON downstream while recording token counts around the same payload.

The worked example begins when an agent/tool workflow sends repetitive structured payloads through a token-billed model path. The principal technical object is a compact lossless exchange payload. Under the normal path, the supported data round-trips through local conversion and the task remains equivalent. This gives the integration a bounded proposition it can test and retain.

The foundation joins Nexus agent messages, Aura MCP tools, retrieval and extraction calls, batch jobs, customer APIs, and any model path where payload tokens are billed by volume. Each caller should depend on the declared input/output and manifest, not on hidden internal state. A product can validate, route, combine or present the component result while keeping HEDL artefact identity and limitations intact.

## FOUNDATION

**HEDL**

compact, local, lossless exchange for model and agent data paths

## OBJECT

**compact  
lossless  
exchange  
payload**

The artefact or state a caller consumes.

PUBLISHED WEBSITE  
SOURCES**Hedl open-  
foundation  
page**

Verify current project/release/source status.

## DECISION

**Fit + reproduce**

Inspect mechanism, integrate, fail, verify, and exit.

## PRIMARY BOUNDARY

Fewer tokens do not guarantee lower total cost, latency, or equal model behaviour for every provider. Published percentages remain tied to the documented benchmark datasets, models, and extraction method.

MECHANISM • INPUT CONTRACT

# Define the component input before invoking it

A file, query, module, model, event or value becomes a reproducible input only when its identity, schema, configuration, rights, and comparison scope are declared.

For the example, an agent/tool workflow sends repetitive structured payloads through a token-billed model path. The component call should bind work/purpose, input identity and origin, schema/format, HEDL version, configuration/profile, target/runtime where relevant, resource and security limits, expected output or verdict, and evidence requirements.

Use content hashes for immutable bytes and signed/versioned identities plus effective metadata for state that cannot be reduced to a file. Preserve collection, parsing, normalisation or translation steps that materially affect meaning. If input is a reference, the manifest must make it resolvable under the selected retention and access policy.

Reject malformed, missing, ambiguous, inaccessible, unauthorised, incompatible, over-limit, or unsupported input before producing a normal-looking output. A valid negative result is different from invalid input; a partial parse or collection is different from empty content; an unsupported target is different from a failed supported execution.

## ILLUSTRATIVE HEDL REQUEST

|           |                            |
|-----------|----------------------------|
| work_id   | component-4f9e             |
| purpose   | bounded-evaluation         |
| input     | content-addressed identity |
| schema    | declared format + version  |
| component | HEDL@pinned                |
| config    | profile + target + limits  |
| result    | typed artefact or verdict  |
| evidence  | manifest + raw check       |

Illustrative envelope, not a claimed current API.

## INPUT MUTATION TEST

Change one identity, schema, configuration, target or authority field and verify the result either changes under a new manifest or is rejected explicitly.

## MECHANISM • EXECUTION

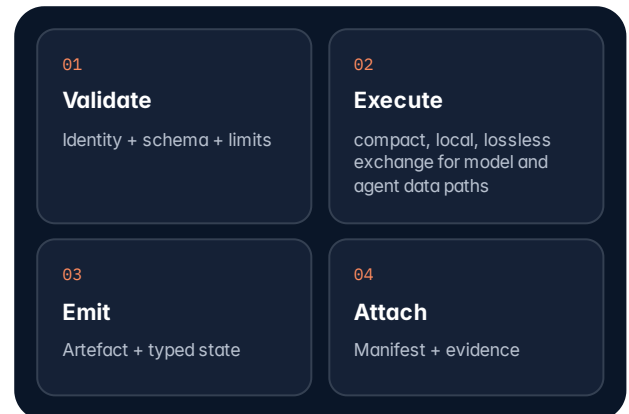
# The HEDL mechanism as a four-stage pipeline

The exact implementation is component-specific; the evaluation pattern remains validate, execute, emit a typed artefact, and bind it to evidence.

HEDL removes syntactic overhead while preserving the supported data semantics and restoring the original exchange shape. Conversion is local and bidirectional; a proxy can speak HEDL upstream and JSON downstream while recording token counts around the same payload.

At receive, validate identity, schema, configuration, capability and limits. At execute, run the declared compact, local, lossless exchange for model and agent data paths mechanism without silently substituting an implementation or profile. At emit, return the artefact plus explicit success, negative, partial, refusal, invalid, failure or unverifiable state. At attach, commit versions, inputs, configuration, raw result and verifier dependencies.

The component's described surfaces are Rust core/crate, CLI, MCP server, and WASM component with local conversion and a zero-runtime-dependency core as described in the published material. Surface adapters can change transport and ergonomics; they should not change the component proposition. Equivalent supported calls through a library, CLI, service, sidecar, FFI, MCP or WASM surface should reconcile to the same input and artefact identities under the declared mode.



A reusable component-evaluation flow for HEDL.

**SURFACE PARITY**

Run one supported case through two selected interfaces and compare semantic result, artefact identity, failure class, and evidence, not only transport status.

MECHANISM • RESULT

# The artefact manifest is part of the output

The caller needs the HEDL artefact and enough identity to interpret, verify, reproduce, migrate, or reject it later.

The material interface includes input/output format and schema, converter version, canonical payload identity, encoded bytes, restored payload, round-trip verdict, token measurement, model/task/dataset context, and interface/runtime identity. The exact object names and schemas come from the current project. The architectural requirement is that the result does not lose the input, component, configuration, target, comparison, and verifier information that gives it meaning.

Return work/result and artefact identity; semantic schema and component versions; input references; configuration/profile; runtime/target; state/error; raw and derived outputs; warnings/limits; integrity or signature; evidence references; and compatibility/retention requirements. Keep human-friendly summary separate from machine-verifiable material.

A downstream product may store only a stable reference when the artefact is sensitive or large. The packet manifest must still resolve the authorised object and make missing, withdrawn, expired or incompatible dependencies explicit. Export should preserve the artefact and manifest, not only a rendered report.

| COMPONENT RECORD                                             |                                          | hedl-artefact |
|--------------------------------------------------------------|------------------------------------------|---------------|
| COMPONENT                                                    | <b>HEDL + version</b>                    | PINNED        |
| INPUT                                                        | <b>identity + schema</b>                 | PINNED        |
| CONFIG                                                       | <b>profile + target + limits</b>         | PINNED        |
| RESULT                                                       | <b>compact lossless exchange payload</b> | TYPED         |
| INTEGRITY                                                    | <b>hash/signature where used</b>         | CHECKED       |
| VERIFIER                                                     | <b>format + runtime + trust</b>          | RETAINED      |
| Illustrative manifest; use the implemented component format. |                                          |               |

## EXPORT TEST

Export a HEDL result and reproduce the declared check using only the documented artefact, manifest, and verifier dependencies.

## MECHANISM • INTERFACES

# Integration surfaces must preserve one semantic contract

HEDL can appear as a library, executable, service, sidecar, FFI, MCP or WASM component; each surface is still the same bounded mechanism.

The published description describes Rust core/crate, CLI, MCP server, and WASM component with local conversion and a zero-runtime-dependency core as described in the published material. For every selected surface, confirm authentication/authorisation where applicable, schema and version, capability discovery, payload/reference limits, memory/time/fuel or resource limits, streaming/order, concurrency, cancellation, errors, retries, idempotency, logs/evidence, packaging, and supported platform/target.

The application should validate the component result before promoting it into product state. Preserve a stable adapter boundary so an implementation can be upgraded or replaced while the caller's semantic contract remains. Provider-specific data belongs in declared extension fields and the manifest, not in hidden behaviour.

Contract tests drive the same corpus through selected surfaces and compare accepted/rejected input, output state, artefact identity, integrity/verdict, resource behaviour, and failure semantics. Transport codes and process exits should map to documented component result classes.

| SURFACE QUESTION  | REQUIRED ANSWER                                   |
|-------------------|---------------------------------------------------|
| <b>Capability</b> | Which operation/version/profile is supported?     |
| <b>Identity</b>   | How are input and artefact named?                 |
| <b>Limits</b>     | Size, time, memory, concurrency, fuel, target?    |
| <b>Errors</b>     | Invalid, negative, partial, failed, unverifiable? |
| <b>Evidence</b>   | Which manifest/raw result/verdict returns?        |
| <b>Packaging</b>  | Which runtime/dependencies/rights are required?   |

The same checklist applies across local and service interfaces.

## ADAPTER RULE

An adapter may translate representation; it must not silently weaken the component's correctness, determinism, security, or evidence contract.

MECHANISM • COMPOSITION

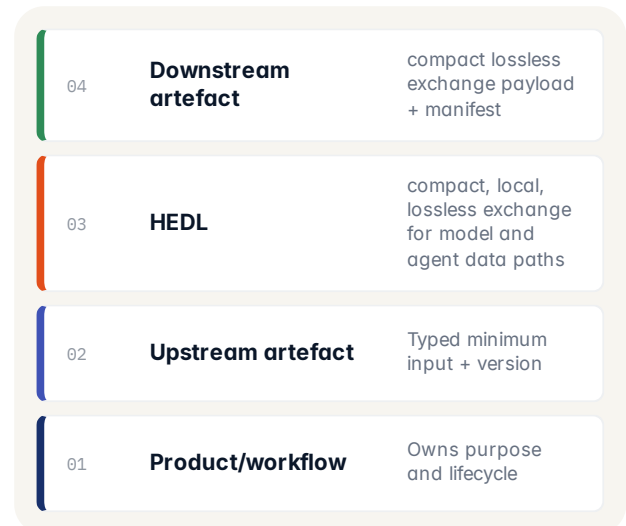
# Place the foundation at the correct stack join

A foundation contributes a narrow artefact to products and other foundations; it does not become a product owner merely because it is reusable.

The relevant joins are Nexus agent messages, Aura MCP tools, retrieval and extraction calls, batch jobs, customer APIs, and any model path where payload tokens are billed by volume. Draw each as upstream input contract → HEDL mechanism → downstream artefact contract. Name the product or workflow owner above it, lower libraries/runtimes it depends on, and the evidence or packet that retains its result.

The upstream caller passes only the minimum input with source, purpose, rights and configuration. The component returns a versioned artefact or reference and state. The downstream caller validates compatibility and may derive new state while preserving lineage. A feedback or update path uses an explicit event/control contract rather than mutating hidden component state.

Mark dependencies required/optional/selected, supported versions, trust/data boundary, resource and determinism effect, packaging and licence, failure mapping, update coupling, and evidence. Then remove or substitute the foundation in a test design to see exactly which promise and object disappear.



A conceptual stack join; exact component dependencies follow the current project manifest.

## NO FOUNDATION SOUP

If several foundations appear in a diagram but no one can name the artefact each contributes, the design is branding rather than architecture.

## MECHANISM • FAILURE

# Failure is part of the component API

The worked fault is: an unsupported value/schema or framing edge enters. The safe result is: conversion refuses or exposes the incompatibility instead of silently changing the payload.

The component-specific failure model is: Unsupported value or schema, ambiguous positional encoding, loss on round trip, ordering/canonicalisation mismatch, proxy framing error, model behaviour change, unavailable downstream JSON consumer, or measurement without task/model context invalidates the “same data” claim.

Expose invalid input, unsupported capability/configuration/target, resource or dependency failure, integrity mismatch, verification failure, valid negative/refusal, and partial result as distinct states. Include stable work/input identity, failing stage, version/configuration, retry safety, partial artefact policy, diagnostic detail, and evidence. Never encode all conditions as empty output.

Recovery can correct input, select an explicitly approved substitute, retry a safe pure operation, resume from checkpoint, rebuild a derived artefact, escalate to an owner, quarantine the result, or stop. If a component can cause external effect, reconcile it before retry. Preserve failed attempts and manifest changes.

| STATE                   | MEANING                               | CALLER ACTION                      |
|-------------------------|---------------------------------------|------------------------------------|
| <b>Invalid</b>          | Input/contract cannot be accepted     | Correct; do not retry unchanged    |
| <b>Unsupported</b>      | Declared capability absent            | Select approved path or stop       |
| <b>Negative/refusal</b> | Valid mechanism result                | Respect outcome/escalate by policy |
| <b>Partial</b>          | Some material unavailable             | Use only if contract permits       |
| <b>Failed</b>           | Execution/dependency/resource error   | Retry/recover by class             |
| <b>Unverifiable</b>     | Integrity/check cannot be established | Quarantine/investigate             |

Exact state names may differ; the semantic distinctions must remain.

## FAULT INJECTION

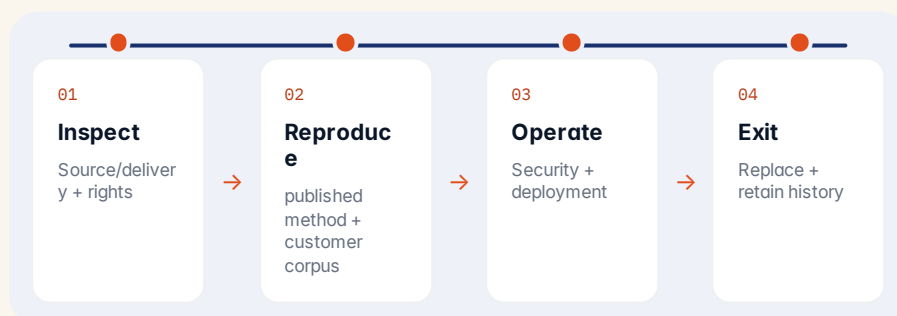
An unsupported value/schema or framing edge enters. Verify that conversion refuses or exposes the incompatibility instead of silently changing the payload.

02

EVALUATION AND ADOPTION

# Reproduce, secure, package, integrate, and exit

A foundation becomes production infrastructure only when its narrow claim survives customer inputs, failure, deployment, operations, change, and removal.



The component decision is evidence-backed and reversible.

EVALUATION • REPRODUCIBILITY

# Reproduce the HEDL claim, not a summary of it

A credible reproduction pins every identity that can affect the declared proposition and retains raw results, limitations, and negative tests.

The canonical exercise is: Round-trip the production payload corpus byte/semantic-equivalently under the documented contract; compare schema edge cases; run direct and proxy paths; reproduce token and accuracy measurements with the named models/tasks; remove the proxy; and confirm the application returns to JSON.

Capture component source or delivered binary identity, build/provenance and runtime environment, dependencies, input/corpus and rights, schemas, configuration/profile, target/hardware/ISA/backend, warm/cold and concurrency conditions where measured, harness/method, repetitions, comparison rule, raw outputs, verdict, reviewer and date.

First reproduce the narrow published claims under its published scope. Then run the customer's representative corpus and target environment. Include malformed, boundary, unsupported, malicious, resource, dependency, integrity and historical cases. A different workload may produce a different performance or quality result without contradicting the original measurement.

| COMPONENT RECORD                                                  |                              | hedl-reproduction |
|-------------------------------------------------------------------|------------------------------|-------------------|
| COMPONENT                                                         | source/binary + version      | PIN               |
| INPUTS                                                            | corpus + identities + rights | PIN               |
| ENVIRONMENT                                                       | target + hardware + runtime  | PIN               |
| METHOD                                                            | harness + comparison         | PIN               |
| RESULT                                                            | raw output + verdict         | RETAIN            |
| LIMITS                                                            | scope + exceptions           | RETAIN            |
| A screenshot or supplier summary is not a reproducibility record. |                              |                   |

## NEGATIVE CONTROL

An unsupported value/schema or framing edge enters. The harness should expose the changed condition rather than still returning a green headline.

## EVALUATION • PERFORMANCE

# Treat every performance number as a measured boundary

Throughput, latency, memory, recall, token reduction, operation coverage, cold start, or energy figures belong to a named workload and method, not the entire platform.

If the HEDL website description publishes a number, preserve its exact component path, corpus or input distribution, size, hardware and instruction set, software version and configuration, warm/cold state, batch/concurrency, repetitions and duration, metric/percentile, accuracy/recall/correctness condition, resource/power method, baseline, exclusions, raw harness and non-SLA qualification.

A fast component can be a small part of end-to-end time; a compact representation can shift work into conversion; an exact result can cost more than an approximate candidate stage; a cold-start advantage can coexist with a slower hot path. Report the full selected path and counter-metrics rather than promoting one route result into universal savings.

Reproduce relied-on numbers as written, then profile the integrated customer workload under steady, burst and degraded conditions. Include preprocessing, transfer, validation, evidence, stores, retries and caller backpressure. Size from service objectives and measured bottlenecks with recovery and growth headroom.

| MEASUREMENT FIELD           | WHY IT MATTERS                     |
|-----------------------------|------------------------------------|
| <b>Workload/corpus</b>      | Defines what was exercised         |
| <b>Hardware/ISA/backend</b> | Defines execution path             |
| <b>Config/warm/batch</b>    | Changes throughput/latency         |
| <b>Correctness/recall</b>   | Prevents speed-only comparison     |
| <b>Baseline boundary</b>    | Prevents unlike systems comparison |
| <b>Raw output/repeats</b>   | Shows distribution and exceptions  |

Use this record for every quantitative claim relied on.

## NO BENCHMARK ALCHEMY

A component benchmark can inform architecture; it cannot become a production SLA, total-cost saving, or whole-workflow outcome without additional evidence.

EVALUATION • SECURITY

# Security includes the component supply chain and parser/runtime boundary

Open or inspectable code helps review; it does not remove build, dependency, input, resource, capability, key, update, and disclosure risks.

Threat-model the HEDL inputs, parsers/decoders or runtime, dependencies, extension/plugin surfaces, FFI or service boundary, resources, files/network where applicable, artefact stores, signatures/keys, verifier, logs, packages, update channel and support/export path. Treat every external input and produced artefact as untrusted until validated.

Request source/repository or delivered binary identity as applicable, licence/rights, build provenance, dependency manifest/SBOM, compiler/toolchain, signatures, vulnerability and disclosure process, unsafe-code posture where stated, sandbox/capability/resource controls where relevant, hardening/fuzz/property tests, and supported platforms/versions.

Test malformed and adversarial inputs, decompression/size/resource exhaustion, path/URL/redirect escape where relevant, parser ambiguity, FFI misuse, service authentication/authorisation, capability denial, modified package, dependency removal, key rotation, verifier abuse, sensitive error/log output, and air-gap closure.

ORIGIN

## Provenance

Source/binary, build, dependencies, signature, licence and release status.

INPUT

## Robustness

Malformed, malicious, huge, ambiguous, private, unauthorised and incompatible cases.

RUNTIME

## Containment

Capabilities, memory/time/fuel, files/network, concurrency and failure isolation.

OUTPUT

## Disclosure/integrity

Schema validation, sensitive logs, artefact access, signing, export and verifier limits.

SECURITY BOUNDARY

Fewer tokens do not guarantee lower total cost, latency, or equal model behaviour for every provider. Published percentages remain tied to the documented benchmark datasets, models, and extraction method.

EVALUATION • DEPLOYMENT

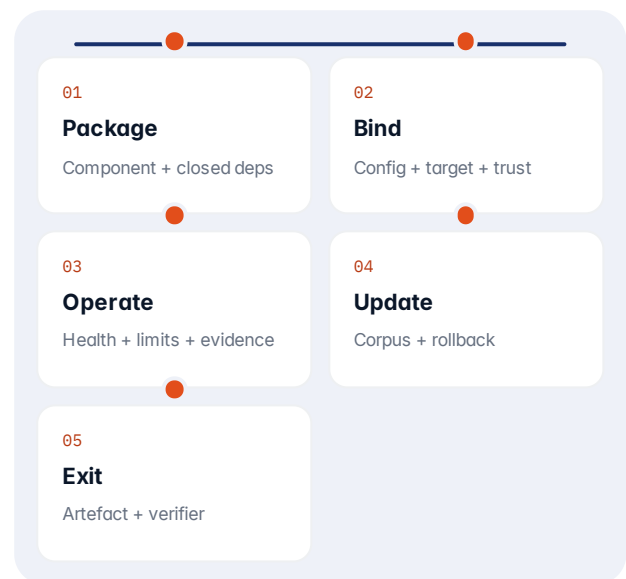
# Package and operate HEDL in the selected posture

A foundation may run embedded, as a local executable, sidecar, service, browser/WASM component or another published form; deployment must preserve its contract.

The published surfaces are Rust core/crate, CLI, MCP server, and WASM component with local conversion and a zero-runtime-dependency core as described in the published material. Build an installed manifest for component, dependencies, runtime/toolchain where required, configuration/profile, platform/CPU features, stores, models/grammars/rules/keys or other data packages, interfaces/ports, limits, health/telemetry, evidence, licence and documentation.

In managed use, define the service and data boundary. In licensed operation, assign customer owners for package, dependency, access, capacity, observability, updates, backup/restore and support. In air gap, close every runtime, licence, trust, verification, update, and diagnostic dependency and validate controlled-media procedures.

Exercise install/cold start, representative load, dependency/resource failure, update and rollback, backup/restore of retained component state, historical artefact verification, no-network operation where required, supplier disconnect, export and decommission. Preserve exact package and environment versions.



Operational lifecycle for HEDL.

## AIR-GAP CHECK

A local executable is not proof of air-gap support if it still requires remote packages, identity, time, keys, licence, telemetry, model, or verification services.

## EVALUATION • INTEGRATION

# Adopt one reversible component path

Start where the foundation replaces a specific opaque, inconsistent, slow, expensive, unsafe, or unverifiable step, and retain the old path until evidence supports the switch.

Choose one real flow involving Nexus agent messages, Aura MCP tools, retrieval and extraction calls, batch jobs, customer APIs, and any model path where payload tokens are billed by volume. Capture current input/output semantics, quality/correctness, latency/resource/cost, errors, operational work, security boundary, evidence and exit. Define what HEDL changes and what the product/workflow owner continues to own.

Integrate behind a versioned adapter; pin component and dependencies; run side-by-side or shadow where possible; compare semantic results and counter-metrics; inject invalid/failure cases; validate artefact/evidence; measure customer hardware and operations; obtain domain/security/technical approval; and cut over segmentally with rollback.

A trace cannot make an external source, model endpoint, human decision or physical device deterministic. It can identify the dependency, capture the response where policy permits, record the version or attestation available and state how replay treats the boundary. Changed or unavailable dependencies are results, not noise to hide.

01

**Read**

inspect published page, source/delivery, licence, interfaces and tests.

02

**Baseline**

Measure current component boundary and failures.

03

**Integrate**

Versioned adapter and minimum data path.

04

**Compare**

Customer corpus, security, performance and evidence.

05

**Cut over**

Segmented, reversible, monitored and owned.

**REVERSIBLE PATH**

At each stage the organisation can remove the component and return to the former contract without losing its data, evidence, or ability to explain work.

EVALUATION • DECISION

# Due diligence, portability, and the exit test

The foundation is production-ready only when mechanism, availability, rights, reproducibility, security, operation, historical verification, and removal are evidenced for the selected use.

Request the current HEDL source or delivered artefact status, repository/release and documentation, licence/rights, supported surfaces/platforms, semantic schemas/formats, dependency and build manifest, benchmark/verification harnesses, security evidence, known limitations, roadmap status, support/update policy, deployment package and compatibility.

Run the component-specific exercise: Round-trip the production payload corpus byte/semantic-equivalently under the documented contract; compare schema edge cases; run direct and proxy paths; reproduce token and accuracy measurements with the named models/tasks; remove the proxy; and confirm the application returns to JSON. Record gaps as verified, not applicable with reason, accepted risk and owner/expiry, contractual condition, remediation and date, or a stop decision. Do not infer “shipping”, “source available”, “MIT”, or product inclusion across components from the open-source overview count.

Binaries, model weights, charts, and every dependency bundle into a signed archive, delivered on encrypted physical media through a documented chain of custody. The import script validates checksums before unpacking. Licence validation is offline-signed. No network egress of any kind after install.

| COMPONENT RECORD                                                    |                                   | hedl-decision |
|---------------------------------------------------------------------|-----------------------------------|---------------|
| MECHANISM                                                           | <b>contract + artefact</b>        | INSPECT       |
| AVAILABILITY                                                        | <b>repo/release/delivery</b>      | VERIFY        |
| RIGHTS                                                              | <b>licence/use/modify/support</b> | CONFIRM       |
| PROOF                                                               | <b>harness + customer corpus</b>  | RUN           |
| OPERATION                                                           | <b>package/update/recover</b>     | PROVE         |
| EXIT                                                                | <b>replace + history verifies</b> | PROVE         |
| A technical selection should retain this decision and its evidence. |                                   |               |

## EXIT EXERCISE

Archive the HEDL artefact, manifest, and verifier dependencies; reproduce the declared check without a Dweve-operated service and record what remains dependent.

## FIND A SUBJECT

# Subject index

Page references point to the substantive explanation, worked example, control, boundary, or decision record, not merely to a passing label.

|                                                           |                                                             |                                                         |
|-----------------------------------------------------------|-------------------------------------------------------------|---------------------------------------------------------|
| <b>A</b> <b>Adoption</b><br>4-5, 13                       | <b>A</b> <b>Air-gapped deployment</b><br>17                 | <b>A</b> <b>Architecture</b><br>11, 15                  |
| <b>A</b> <b>Artefact identity</b><br>6, 8-10              | <b>A</b> <b>Assurance</b><br>3-4                            | <b>A</b> <b>Aura</b><br>6, 11, 18                       |
| <b>A</b> <b>Authority</b><br>7, 10, 16                    | <b>A</b> <b>Availability</b><br>4, 19                       | <b>B</b> <b>Benchmarking</b><br>3-4, 6, 15-16, 19       |
| <b>B</b> <b>Boundaries</b><br>4, 6, 10-11, 14-18          | <b>C</b> <b>Capabilities</b><br>8, 10, 12, 16               | <b>C</b> <b>Compliance</b><br>4                         |
| <b>C</b> <b>Configuration</b><br>4, 7-9, 11-12, 14-15, 17 | <b>C</b> <b>Contracts</b><br>3, 5-7, 10-12, 14, 17-19       | <b>C</b> <b>Cost and budgeting</b><br>3, 6, 15-16, 18   |
| <b>D</b> <b>Dependencies</b><br>5, 8-12, 14, 16-19        | <b>D</b> <b>Deployment</b><br>3, 13, 17, 19                 | <b>D</b> <b>Determinism</b><br>4, 10-11, 18             |
| <b>D</b> <b>Due diligence</b><br>19                       | <b>E</b> <b>Energy</b><br>15                                | <b>E</b> <b>Evidence</b><br>3, 5, 7-13, 15, 17-19       |
| <b>E</b> <b>Exit</b><br>3, 6, 13, 17-19                   | <b>F</b> <b>Failure</b><br>5, 8, 10-13, 16-18               | <b>H</b> <b>Hardware</b><br>4, 14-15, 18                |
| <b>H</b> <b>HEDL</b><br>3-11, 14-19                       | <b>I</b> <b>Identity</b><br>5-10, 12, 14, 16-17             | <b>I</b> <b>Inputs</b><br>4-12, 14-16, 18               |
| <b>I</b> <b>Integration</b><br>3-4, 6, 10, 18             | <b>K</b> <b>Keys and signing</b><br>9, 16                   | <b>L</b> <b>Ledger</b><br>4, 9, 14, 19                  |
| <b>L</b> <b>Licensing</b><br>3-4, 11, 16-19               | <b>L</b> <b>Limits and exclusions</b><br>4, 7-10, 14, 16-17 | <b>L</b> <b>Lineage</b><br>11                           |
| <b>M</b> <b>Manifests</b><br>4-12, 16-17, 19              | <b>M</b> <b>Migration</b><br>4                              | <b>M</b> <b>Models and solvers</b><br>3-9, 11-12, 16-19 |

## SUBJECT INDEX CONTINUED

# Subject index · N–W

Terms are grouped alphabetically. Consecutive page references are collapsed into ranges.

|                                                     |                                                         |                                                         |
|-----------------------------------------------------|---------------------------------------------------------|---------------------------------------------------------|
| <b>N</b> <b>Nexus</b><br>6, 11, 18                  | <b>O</b> <b>Operations</b><br>10, 12, 15, 17, 19        | <b>O</b> <b>Outputs</b><br>5–7, 9–10, 12, 14–16, 18     |
| <b>O</b> <b>Ownership</b><br>11–12, 18–19           | <b>P</b> <b>Performance</b><br>14–15, 18                | <b>P</b> <b>Policies</b><br>7, 12, 18–19                |
| <b>P</b> <b>Portability</b><br>19                   | <b>P</b> <b>Proof</b><br>3, 17, 19                      | <b>P</b> <b>Provenance</b><br>4, 14, 16                 |
| <b>R</b> <b>Recovery</b><br>12, 15, 17              | <b>R</b> <b>Replay</b><br>18                            | <b>R</b> <b>Reproducibility</b><br>3, 6, 9, 13–15, 19   |
| <b>R</b> <b>Responsibility</b><br>6                 | <b>R</b> <b>Retention</b><br>7, 9                       | <b>R</b> <b>Rights</b><br>3–4, 7, 10–11, 13–14, 16, 19  |
| <b>R</b> <b>Risk</b><br>19                          | <b>R</b> <b>Rules</b><br>10, 14                         | <b>S</b> <b>Security</b><br>3, 7, 10, 13, 16, 18–19     |
| <b>S</b> <b>Service levels</b><br>4, 15             | <b>S</b> <b>Sources</b><br>3–4, 6, 11, 13–14, 16, 18–19 | <b>S</b> <b>Sovereignty</b><br>3, 6, 8, 12, 16, 18      |
| <b>S</b> <b>State</b><br>6–12, 15, 17–18            | <b>S</b> <b>Supply chain</b><br>16                      | <b>T</b> <b>Testing</b><br>6–7, 9, 11, 16, 19           |
| <b>T</b> <b>Trace</b><br>18                         | <b>V</b> <b>Validation</b><br>15–16, 19                 | <b>V</b> <b>Verification</b><br>3–4, 6–7, 9, 12, 17, 19 |
| <b>V</b> <b>Versioning</b><br>4, 7, 9–12, 14–15, 18 | <b>W</b> <b>Workflows</b><br>3–4, 6–7, 11, 15, 18       |                                                         |

## CLOSING COMPONENT RECORD

# The next HEDL action is an independent component proof

Use the real input, configuration, target and failure that matter to the selected workflow. Preserve the artefact and enough dependencies that another authorised team can repeat the check.

Every AI call is priced by the token, and much of that budget is syntax rather than data. HEDL sent the same request in 56 percent fewer tokens in the measured run and showed a +10.3pp accuracy difference from JSON there. That is evidence from this setup, alongside a lower bill, without changing downstream systems.

Run the primary exercise: Round-trip the production payload corpus byte/semantic-equivalently under the documented contract; compare schema edge cases; run direct and proxy paths; reproduce token and accuracy measurements with the named models/tasks; remove the proxy; and confirm the application returns to JSON. Include the negative case, an unsupported value/schema or framing edge enters, and confirm that conversion refuses or exposes the incompatibility instead of silently changing the payload. Capture source/binary, dependencies, inputs/corpus, configuration, target/hardware, method, raw output, verdict, security/failure results and limitations.

Then package and operate it in the intended posture, update and roll back, restore any retained state, verify a historical artefact, export and replace the component, and record what still depends on Dweve. Decide proceed, narrow, remediate, accept with owner/expiry, contract a condition, or stop.

| COMPONENT RECORD                                           |                                           | hedl-proof |
|------------------------------------------------------------|-------------------------------------------|------------|
| CONTRACT                                                   | <b>input + output + failure</b>           | INSPECT    |
| ARTEFACT                                                   | <b>identity + manifest</b>                | RETAIN     |
| REPRODUCE                                                  | <b>published method + customer corpus</b> | RUN        |
| SECURITY                                                   | <b>abuse + supply chain</b>               | RUN        |
| OPERATION                                                  | <b>package/update/recover</b>             | PROVE      |
| EXIT                                                       | <b>replace + history</b>                  | PROVE      |
| A component proof ends in an auditable selection decision. |                                           |            |

### BOUNDARY

Fewer tokens do not guarantee lower total cost, latency, or equal model behaviour for every provider. Published percentages remain tied to the documented benchmark datasets, models, and extraction method.