A Provenance-Routed Regenerative Object Codec

The fold/2 P/D/M/R/G representation system with exact proof and fallback

Hayden Lindley · Glyphd Labs

2026-07-27

Contents

A Provenance-Routed Regenerative Object Codec

Abstract

General-purpose compression assumes that an object is primarily a byte sequence. Many modern objects also have recoverable context: a prior version, a shared dictionary, an exact source, a deterministic generator, a model that emitted the sequence, a procedural recipe, or a lineage graph. Treating every such object as context-free bytes discards information that can be used for storage, transmission, regeneration, and random access. Treating generation as proof, however, risks silent drift and dependency laundering.

Object-Oriented Compression (OOC), serialized as `fold/2`, is a proof-bound meta-codec for objects with provenance. It selects among five routes:

- **P:** proven conventional fallback;
- **D:** dictionary/reference/delta conditioned;
- **M:** model-conditioned lossless;
- **R:** exact deterministic regeneration;
- **G:** regeneration plus exact residual.

Every route declares dependencies, accounting mode, exact content identity, representation identity, sealed layers, proof, and fallback. The encoder must decode and digest-verify its own output before sealing. The decoder fails closed or follows a declared fallback ladder. A spatial address may guide locality but never determines identity.

The current proof deck establishes several mechanisms: exact arithmetic and text round trips, boundary verification, corruption refusal, conditioned sample behavior, recipe regeneration, exact residual correction, and seed-drift sensitivity. It does not prove a new general-purpose compressor. The strongest research target is narrower: generated, versioned, related, and structured objects for which provenance creates recoverable redundancy.

1. Compression should know what an object is

A conventional codec sees bytes and perhaps a media type. OOC also asks:

- Was this object generated from a recipe?
- Is there a prior accepted version?
- Does the project maintain a shared dictionary?
- Did a language model already compute probabilities while emitting it?
- Can a nearby canonical object serve as a reference?
- Which exact sources are required to reconstruct it?
- What does a clean machine need?
- What happens if a dependency disappears or drifts?

These questions do not make the object smaller by themselves. They identify candidate routes that may exploit existing structure.

The meta-codec does not insist on novelty. If no special route wins after honest accounting, the object uses the best approved ordinary codec.

2. Identity law

`fold/2` separates:

- **ContentId:** exact canonical bytes;
- **LogicalId:** object across versions;
- **RepresentationId / fold_digest:** one encoded fold;
- **SpatialAddress:** optional placement hint;
- **dependency identities:** exact prerequisites;
- **execution identity:** pinned regeneration environment.

This prevents two dangerous substitutions:

1. a compact recipe being treated as the exact object before regeneration verifies;
2. a spatial address or semantic class being treated as content identity.

The proof binds `content_id` to `fold_digest`. Corrupting content, representation, or dependency breaks the binding.

3. FoldV2

A public-safe object model is:

```
FoldV2 {
  format_version
  content_id
  canonical_size
  media_type
  provenance_class
  route
  context_manifest?
  execution_capsule?
  layers[]
  fallback_ref?
  spatial_hint?
  proof
}
```

A `SealedLayer` records:

```
layer_id
purpose
codec
uncompressed_len
compressed_len
digest
independent_decode
code_bytes
byte_range
dependencies[]
```

The format requires deterministic serialization, bounded lengths, version handling, unknown-field behavior, and cryptographic digest identifiers.

Progressive disclosure occurs only at independently verified boundaries. An arithmetic stream prefix is not trusted merely because it decodes into plausible text.

4. Encoder law

The encoder must fully decode and digest-verify its own output through the normative decoder before sealing proof.

> An unverified fold is not a fold. It is a draft.

This catches:

- corrupt code streams;
- missing dependencies;
- wrong dictionary order;
- nondeterministic regeneration;
- residual mismatch;
- serializer divergence;
- length errors;
- fallback failure.

The law is expensive compared with writing bytes once. The cost is appropriate because the representation is making an exact recoverability claim.

5. Route P — proven fallback

Route P stores the object using the best approved conventional representation:

- raw;
- zstd;
- Brotli;
- xz;
- lossless media codec;
- another mature job-specific codec.

It has no unusual dependency requirement.

P provides the global invariant:

> No object should become materially larger merely because it entered the OOC system.

The router may include a manifest floor or small fixed overhead. If an experimental route loses beyond the declared tolerance, P wins.

6. Route D — reference conditioned

D uses known exact context:

- prior version;
- shared dictionary;
- exact source;
- content-defined chunks;
- project dictionary;
- parent or neighbor;
- structured delta;
- VCDIFF/bsdiff-style base.

Every conditioning source appears in a `ContextManifest` with canonical ordering and exact identity.

D must be compared with strong conditioned baselines, not unconditioned gzip. A reference route that “wins” only because the baseline was denied the same prior version is invalid.

D is likely the first production-worthy research route because it uses conventional mechanisms under stronger provenance discipline.

7. Route M — model-conditioned lossless

A pinned model supplies token or byte probabilities to an entropy coder. The model, tokenizer, runtime, probability quantization, context, and precision become dependencies.

Cold-start accounting includes those dependencies. A tiny code stream cannot pretend that a multi-gigabyte model costs zero.

M-flight

The flagship M profile captures probabilities during generation:

```
ModelCodedSegment {
  model_execution_ref
  context_horizon_digest
  tokenizer_or_byte_contract
```

```
  probability_quantization_contract
  code
  exact_output_digest
  raw_fallback_ref?
}
```

The generating model already computed the distribution. Capturing the emitted token under that distribution avoids a second inference pass.

The difficult part is decoder conformance. Encoder and decoder must derive bit-identical integer CDFs. Runtime, hardware, kernels, precision, and quantization can change probabilities. If conformance cannot be guaranteed, the segment cannot be sealed as exact M-flight.

The first target is not arbitrary human text. It is model-generated structured output under a pinned local runtime:

- SurfaceDeltas;
- agent logs;
- build packets;
- meeting decisions;
- source-backed summaries;
- typed journals.

8. Route R — exact regeneration

R stores a compact `ExecutionCapsule` that deterministically regenerates the object.

A capsule may include:

```
generator graph digest
model/weights digest
tokenizer/VAE/sampler digests
runtime/container digest
deterministic kernel policy
precision
hardware compatibility class
parameters
seeds
input refs
test vector
expected content digest
```

R is valid only if regeneration at encode time produces the exact `ContentId`.

A seed and model name alone are insufficient. Numerical kernels, library versions, hardware, and stochastic detail can alter output.

R is the special case in which the residual is empty.

9. Route G — regeneration plus residual

G regenerates an approximate candidate and stores an exact residual.

```
canonical bytes
= regenerate(capsule)
+ exact correction(residual)
```

The representation remains lossless if the residual application and final digest verify.

G is likely more robust for generated media than R. A pinned generator may reproduce structure while differing in noise, grain, or floating-point detail. The residual corrects the candidate to exact output.

The proof deck demonstrates that pinning stochastic detail can materially reduce residual size and that changing a seed can destroy the advantage. This is a mechanism result on a synthetic scene, not a general media claim.

10. Fallback ladder

Decode follows:

```
R conformance fails
→ use G residual if present
→ use fallback_ref if present
→ refuse with explicit error
```

The decoder must not silently return an approximate object in exact mode. A failed regeneration is an integrity event.

Write-ahead raw retention can protect against crashes while an experimental route is sealing. The raw fallback can be garbage-collected only after proof and policy permit.

11. Context shelves

A project may expose hierarchical context candidates:

- global dictionary;
- project dictionary;
- branch dictionary;
- prior version;
- parent object;
- selected exact neighbors;
- generator prior.

The encoder trial-encodes candidates and records the winner. FOVEA may suggest nearby contexts, but moving an object cannot change its `ContentId` or canonical address.

Context selection must avoid train/test leakage in benchmarks. A dictionary built from the test object invalidates the result.

12. Accounting

OOC reports three sizes:

1. **object bytes:** bytes stored uniquely for the object;
2. **amortized recoverability bytes:** object bytes plus allocated shared dependencies;
3. **cold-start recoverability bytes:** everything required to reconstruct on a clean machine.

It also reports:

- encode/decode time;
- peak memory;
- energy where measured;
- random-access cost;
- dependency fetch;
- fallback rate;

- exactness;
- conformance failures.

A 68-byte recipe with a 4 GB dependency is not a 68-byte cold-start object.

13. Current mechanism evidence

The recorded proof-deck run passed twelve of twelve self-tests:

- arithmetic-code round trip;
- text-fold round trip;
- address/Hilbert invariants;
- gist boundary verification;
- corruption refusal;
- neighborhood-conditioned rate reduction on the supplied sample;
- recipe regeneration;
- conditioned residual reconstruction;
- seed-drift failure.

The Rust sources contain broader suites for FOVEA, origami codec, and OOC-video. In the assessment environment, those were inspected but not executed.

The evidence establishes:

- a legitimate progressive conditioned container mechanism;
- exact corruption detection in the demo;
- recipe and residual paths;
- meaningful dependency sensitivity.

It does not establish superiority to mature codecs on general text or video.

14. Benchmark laboratory

A run receipt includes:

```
commit
codec_config
route
corpus_digest
train_split_digest
dependency_digest
environment_digest
object_bytes
amortized_bytes
cold_start_bytes
encode_ms
decode_ms
peak_memory
energy
random_access
exactness
fallback
```

Corpora

Report separately:

- ordinary captured text;

- source code;
- versioned documents;
- agent logs;
- structured model output;
- generated images;
- generated video;
- procedural assets;
- highly related object families.

Baselines

Use job-specific opponents:

- zstd and zstd dictionary;
- Brotli;
- xz;
- VCDIFF/bsdiff;
- dedup/content-defined chunks;
- FFV1 or lossless media codecs;
- AV1/AV2 only under appropriate lossy or exact contracts;
- raw fallback.

Stop-line gates

- exactness failure;
- corruption accepted;
- dependency omitted from accounting;
- test leakage;
- hidden model cost;
- unrecoverable object without declared refusal;
- benchmark against a weaker class.

15. Security

A decoder executes complex representations and possibly generators. Threats include:

- decompression bombs;
- recursive dependency bombs;
- malicious capsules;
- untrusted containers;
- path traversal;
- resource exhaustion;
- model supply-chain compromise;
- digest downgrade;
- ambiguous serialization;
- fallback substitution;
- poisoned dictionaries.

Controls include strict limits, allowlisted codecs/runtimes, sandboxed execution, cryptographic digests, deterministic serialization, dependency depth caps, time/memory budgets, signatures where appropriate, and refusal on unknown proof algorithms.

16. Limitations

The meta-codec adds metadata, trial encoding, dependencies, and complexity. Ordinary data may receive no benefit. Model-conditioned exact decoding may be impractical across hardware. Exact deterministic regeneration of neural media can be fragile. Residuals can approach full size. Shared dependencies complicate portability.

The "never materially worse" invariant requires disciplined router thresholds and cannot erase the fixed manifest cost for tiny objects.

The current evidence is mechanism-level and synthetic in important places. Real generated-media and large-corpus results remain to be run.

17. Conclusion

OOC is not "fractal compression." It is a provenance-aware route selector.

Its strongest promise is:

> Store ordinary bytes ordinarily; store related bytes against declared recoverable context; store generated bytes as verified execution plus exact correction when that wins.

The codec earns trust by preserving fallback, accounting for dependencies, and refusing silent approximation. Its ambition is high, but its law is conservative: no compact representation is allowed to become a proof unless it can reconstruct the exact object it claims to represent.

Architecture illustration briefs

- **P10-F01 — Five-route meta-codec**: Router branches to P, D, M, R, and G, each with dependencies and all converging on normative decode, digest verification, proof, and fallback.
- **P10-F02 — fold/2 object envelope**: Fold manifest with exact content ID, representation digest, context manifest, execution capsule, sealed layers, proof, and fallback.
- **P10-F03 — Recipe plus residual**: Generator produces candidate; residual corrects; final digest compares with canonical bytes; mismatch follows fallback/refusal.
- **P10-F04 — Honest byte accounting**: Nested bars for object bytes, amortized dependencies, and cold-start recoverability, contrasting misleading recipe-only reporting.

Source register

- **P10-S01** — GlyphDrive OOC fold/2 Specification v0.2 (2026-07-12). `GLYPHDRIVE-OOC-FOLD-SPEC-v0.2.md`
- **P10-S02** — GlyphDrive/FOVEA/OOC Technical Assessment (2026-07-12). `GLYPHDRIVE-FOVEA-OOC-TECHNICAL-ASSESSMENT-v0.2-PLANNING.md`
- **P10-S03** — GlyphDrive Codec Benchmark Contract (2026-07-12). `GLYPHDRIVE-CODEC-BENCHMARK-CONTRACT-v0.1.md`
- **P10-S04** — GlyphDrive v0.2 Ratification Gate R0 (2026-07-12). `PASTE-INTO-OPUS-GLYPHDRIVE-V02-RATIFICATION-R0.md`
- **P10-S05** — GlyphCAS Design Report (2026-07-24). `How_we_can_make_CAS_lightning_fast_for_any_LL`

Public-disclosure and IP boundary

This edition publishes the public-safe route architecture, format obligations, mechanism evidence, benchmark doctrine, and limitations. It excludes restricted codec heuristics, confi-

dential corpus details, unreleased optimizations, and patent-claim drafting. No general compression or AV1-superiority claim is authorized.