GlyphCAS

Exact content, typed semantic identity, and evidence-preserving reuse for language-model systems

Hayden Lindley · Glyphd Labs

2026-07-27

Contents

GlyphCAS

Abstract

Conventional content-addressable storage is exact by design: identical bytes receive identical content identities; different bytes do not. Language-model systems often want a second property: repeated meanings, structures, plans, or outputs should be reusable even when their surface bytes differ. A naive “semantic hash” can create speed, but it can also erase provenance, collapse task-specific differences, cross project or authority boundaries, and promote approximate similarity into false exactness.

GlyphCAS is a layered identity and reuse architecture that keeps exact content identity sovereign while adding typed, contract-bound derived identities. It separates content, semantic, contextual, dependency, representation, logical, and receipt identities. A model may propose a semantic graph or reuse candidate, but deterministic canonicalization, schema validation, authority checks, versioned compilers, and explicit Equivalence Contracts decide whether an alias may be minted or used.

The reference package implements exact SHA-256 storage, deterministic canonical JSON over a bounded subset, Unicode normalization, content-addressed semantic identities, contextual reuse keys, controlled paraphrase snapping, project and authority isolation, dependency invalidation, hash-chained receipts, `fold/2` representation objects, exact recipe verification, prefix/KV cache keys, deterministic software routing, and ten passing unit tests.

The current evidence supports a reference architecture and narrow mechanism claims. It does not establish universal semantic canonicalization, universal model-independent cache reuse, production latency gains, or a general-purpose compression result.

1. Exact CAS is a truth primitive

A cryptographic content identifier answers a narrow and powerful question:

> Are these exact bytes the same?

That question is useful precisely because the answer does not depend on interpretation. It supports integrity, deduplication, immutable artifacts, reproducible dependencies, Merkle structures, and proof.

An AI system should not weaken this property in the pursuit of “meaning.” Two paraphrases may be equivalent for answering a FAQ and dangerously different for:

- exact quotation;
- legal interpretation;
- medication instructions;
- code patching;
- transaction authorization;
- a user's personal voice;
- provenance;
- policy;
- safety boundaries.

Therefore, GlyphCAS begins with a constitutional rule:

> **Exact bytes remain sovereign proof.**

A semantic class can point to exact objects. It cannot replace their exact identity.

2. The identity stack

GlyphCAS uses several linked identities.

2.1 ContentId

A cryptographic digest of exact bytes or a deterministic exact object serialization.

```
content_id = H(exact_bytes)
```

A `ContentId` is valid for exact retrieval and integrity.

2.2 LogicalId

The durable identity of an object across versions:

```
project:P7/decision:D42
artifact:A9
actor:maya
```

The current exact payload of a logical object may change while the logical identity remains.

2.3 SemanticId

A cryptographic digest of a deterministic canonical semantic graph under a declared compiler and profile:

```
semantic_id =
  H(canonical_semantic_graph
    + semantic_profile
    + compiler_version)
```

A `SemanticId` means that the compiler produced the same declared semantic class. It does not mean that the input bytes were identical.

2.4 ContextualReuseKey

A reuse key binds the semantic class to the conditions under which reuse is allowed:

```
reuse_key =
  H(semantic_id
    + project_scope
    + authority_scope
    + task_contract
    + policy_version
    + freshness
    + dependency_root
    + model_or_runtime_profile)
```

Changing project, authority, policy, dependencies, or task can invalidate reuse even when the semantic class remains.

2.5 DependencyId

A digest or Merkle root over exact dependencies, versions, ordering, and role.

2.6 RepresentationId

The identity of one encoding, cached form, tokenization, KV artifact, `fold/2` object, or generated representation. Several representations may resolve to the same content.

2.7 ReceiptId

The identity of a recorded operation: store, hit, miss, compile, snap, invalidate, regenerate, verify, or refuse.

These identities create an explicit graph rather than one overloaded “smart hash.”

3. Canonicalization is bounded and versioned

Semantic canonicalization is not a magical ability to discover one true meaning. It is a compiler operating under a contract.

A semantic compiler may produce typed nodes such as:

```
claim
constraint
authority
condition
substitution
uncertainty
action
source
object
relationship
```

The model may help transform language into this graph. Trust begins only after:

- schema validation;
- deterministic normalization;
- canonical ordering or labeling;
- compiler version binding;
- source and project scope;
- authority checks;
- ambiguity handling;
- prohibited-substitution checks.

A canonical serializer must define Unicode normalization, key ordering, number handling, unknown fields, limits, and version behavior. The reference subset rejects floats to avoid cross-runtime ambiguity and uses deterministic JSON conventions for supported objects.

A compiler change creates a different dependency. Old and new semantic identities may be related through migration, but they cannot be silently treated as identical.

4. Equivalence Contracts

Similarity is not enough. Reuse requires an **Equivalence Contract**.

A contract declares:

```
contract_id
task_family
preserved_invariants[]
allowed_variation[]
```

```
prohibited_substitutions[]
authority_scope
project_scope
freshness
exact_evidence_required
approximation_allowed
verifier_id
max_age
```

Three modes remain separate.

EXACT_CONTENT

The exact input content is identical. The lookup includes a `ContentId`. No paraphrase snapping occurs.

EXACT_SEMANTIC

The deterministic compiler produced the same semantic class under the same contract and dependencies. The system may retrieve exact stored output bytes associated with the alias, but it labels the event as a semantic reuse hit rather than an exact input match.

APPROXIMATE

A candidate falls within declared task-specific tolerances. An explicit verifier checks preserved invariants. The result is labeled `APPROXIMATE_VERIFIED`; it is never promoted to exact CAS identity.

This structure allows a system to be aggressive about reuse without becoming dishonest about sameness.

5. The two linked DAGs

The original semantic Merkle intuition becomes safer when split into two linked graphs.

Exact Content DAG

The exact layer uses:

- cryptographic chunks;
- content-defined boundaries where appropriate;
- exact parent links;
- immutable blobs;
- exact object roots.

Content-defined chunking can localize changes so insertions do not shift every downstream boundary. Exact leaves remain valid even when their conceptual role changes.

Semantic Object DAG

The semantic graph represents conceptual units:

- claims;
- code symbols;
- visual objects;
- constraints;

- project objectives;
- reusable macros;
- source-backed statements;
- branches;
- work orders.

Each semantic node links to exact `ContentId` values that support it.

```
SemanticNode:
  semantic_id
  type: background_texture
  exact_refs:
    - content:chunk_14
    - content:chunk_15
  relations:
    - BEHIND foreground_object
    - SATISFIES style_anchor
```

Semantic structure can change without rewriting unrelated exact chunks. Exact chunks can deduplicate even when their semantic roles differ.

6. Layered early exits

GlyphCAS aims for speed through a sequence of increasingly expensive exits, not one universal semantic hash.

A possible route is:

1. exact byte hit;
2. normalized exact-input hit under a declared normalization;
3. exact chunk/Merkle reuse;
4. cached Glyph compilation;
5. deterministic macro expansion;
6. exact semantic alias under unchanged context;
7. context-specific recipe or prior-version reuse;
8. FOVEA active-neighborhood retrieval;
9. stable PromptCapsule and prefix-cache reuse;
10. model-specific token/KV cache;
11. verified recipe regeneration;
12. full inference.

Each stage must be cheaper than the remaining path and must preserve the required invariants. A semantic compiler that costs more than the avoided inference can lose on latency even if its reuse decision is correct.

The architecture is therefore benchmark-driven. No fast path receives immunity.

7. Model-neutral and model-specific artifacts

“For any LLM” cannot mean that one model's internal cache is portable to every other model. GlyphCAS divides artifacts into two classes.

Model-neutral

- exact blobs;
- source spans;
- semantic graphs;

- Equivalence Contracts;
- PromptCapsules;
- dependencies;
- Work Orders;
- Returns;
- Receipts;
- deterministic macros;
- accepted project state.

Model-specific

- tokenized spans;
- prefix blocks;
- KV cache;
- attention-state fragments;
- quantized cache blocks;
- execution capsules;
- tokenizer-dependent probability streams.

Changing the model invalidates the model-specific lane while preserving project truth and model-neutral semantics. This is the honest basis for provider independence.

8. Semantic uncertainty as a gate

Semantic identity is inferred. Uncertainty must not disappear merely because a cache hit is convenient.

A compiler can emit:

```
candidate_semantic_id
uncertainty
ambiguities[]
contradictions[]
required_verification
prohibited_fast_paths[]
```

Routing then considers:

```
authority criticality
uncertainty
action risk
evidence requirement
dependency health
freshness
downstream consequence
```

A low-risk formatting request may reuse a semantic alias. A legal quotation, security action, or high-authority substitution should route to exact evidence and stronger verification.

This turns semantic uncertainty into an operational signal rather than a hidden similarity threshold.

9. SurfaceDelta and local invalidation

Regenerating a complete semantic object can invalidate large caches. GlyphCAS adopts `SurfaceDelta`-style patches:

```
base_version
operations[]
authorizing_token_ids[]
evidence_refs[]
expected_invariants[]
```

The validator applies the patch or rejects it. Only changed semantic nodes, exact chunks, PromptCapsule tails, and model-specific cache spans are invalidated.

This makes the mutation unit smaller, auditable, and more cacheable.

10. Glyph Macros and recursive reuse

Repeated subgraphs can become deterministic macros:

```
macro_id
pattern
parameters
expansion_version
exact_expansion_digest
dependency_refs
```

A macro is not trusted merely because it is frequent. It requires deterministic expansion and limits on recursion, size, and dependency depth.

This provides a bridge from repeated language or graph motifs to exact reuse without claiming that geometry computes meaning.

11. Representation routes

GlyphCAS can link exact and semantic identities to `fold/2` representation routes:

- **P:** mature codec or raw fallback;
- **D:** dictionary, prior version, chunk, or exact reference;
- **M:** model-conditioned lossless;
- **R:** exact deterministic recipe;
- **G:** recipe plus exact residual.

A representation object retains:

```
content_id
representation_id
route
dependencies
execution_capsule?
context_manifest?
layers
proof
fallback
```

The representation may be replaced without changing exact content identity. Regeneration must digest-match or fall back/refuse.

12. Receipts and invalidation

Every fast path emits a receipt. A hit should be more explainable than “cache returned value.”

A receipt records:

```
request_identity
lookup_mode
content_id?
semantic_id?
contextual_reuse_key?
contract_id
dependency_root
project_scope
authority_scope
compiler_version
model_profile
result_content_id
verifier_result
latency
fallback_taken
```

Reverse dependency indexes allow targeted invalidation:

- source changes;
- policy changes;
- authority revocation;
- compiler update;
- model/runtime change;
- contract expiration;
- semantic graph correction.

Immutable exact blobs need not be deleted when an alias becomes invalid. The system removes or supersedes the reuse path.

13. Reference implementation evidence

The accompanying Python reference package implements and tests:

- exact SHA-256 blob CAS;
- deterministic canonical JSON over a bounded subset;
- Unicode normalization;
- no-float canonicalization rule;
- content-addressed semantic identities;
- contextual reuse keys;
- controlled semantic snapping across defined paraphrases;
- exact-content and exact-semantic mode separation;
- project and authority isolation;
- dependency-indexed invalidation;
- hash-chained receipts;
- P/D/M/R/G representation objects;
- exact recipe verification;
- model-specific prefix and KV cache keys;
- deterministic software scheduling;
- ten passing unit tests.

This evidence supports specific mechanism statements. It does not prove the broad design target “lightning fast for any LLM.”

14. Benchmark program

Correctness gates

- zero silent exact-content corruption;
- zero semantic candidate promoted to exact content identity;
- zero cross-project reuse without an accepted contract;
- zero explicitly blocked substitution;
- deterministic canonicalization for supported inputs;
- deterministic macro expansion;
- dependency invalidation correctness;
- exact citations verified or flagged;
- authority and policy changes break reuse;
- regeneration digest mismatch fails closed.

Performance

Measure:

- exact hit rate;
- semantic hit precision;
- false reuse;
- compiler overhead;
- end-to-end latency;
- time to first useful output;
- prompt tokens and prefill;
- model calls avoided;
- bytes and dependency-adjusted bytes;
- cache footprint;
- invalidation cost;
- cold-start recoverability.

Quality

Measure:

- task success;
- structured-output validity;
- repairs;
- evidence completeness;
- staleness;
- branch reopening accuracy;
- action traceability.

Every result must be reported per adapter and workload. One model, one task family, or one semantic profile cannot justify a universal claim.

15. Security

The fast path creates attack surfaces:

- authority spoofing;
- semantic alias poisoning;
- cross-project leakage;
- stale anchor reuse;
- malicious macro dictionaries;

- graph bombs;
- recursive expansion;
- source-span tampering;
- prompt injection encoded as high-authority tokens;
- model attempts to change compiler policy;
- approval replay;
- cache poisoning;
- hidden substitutions;
- forged recipe dependencies.

Controls include:

- typed but untrusted inputs;
- project and authority scope in keys;
- signed/versioned compilers;
- immutable exact sources;
- recursion and graph limits;
- dependency and policy versioning;
- explicit approval for consequential reuse;
- exact evidence retrieval;
- quarantine;
- receipts;
- fail-closed regeneration.

A typed semantic node is not trusted because it is structured.

16. Limitations

Semantic canonicalization remains a hard inference problem. Two inputs can be equivalent under one task and different under another. A compiler can be nondeterministic or biased. Canonical graph labeling can be expensive. Semantic aliases can reduce model calls while increasing total latency if compilation dominates.

Model-specific KV reuse is highly dependent on tokenizer, weights, runtime, positional encoding, cache layout, attention implementation, precision, and API support. The core architecture can store the contracts but cannot promise portability.

Content-defined chunking and Merkle DAGs add metadata and may not help small objects. Macro reuse can create dependency complexity. `fold/2` routes can lose against mature codecs after honest dependency accounting.

The reference package is intentionally narrow. Production scale, concurrency, distributed consistency, multi-tenant isolation, and adversarial security require independent engineering.

17. Conclusion

Exact CAS and semantic reuse should not be forced into one identity.

GlyphCAS treats speed as a ladder of verified exits. Exact bytes remain the final proof. Semantic identities are useful indexes created under explicit contracts. Context, authority, dependencies, model version, and task determine whether reuse is lawful.

The design principle is:

Make approximation useful without allowing it to impersonate exactness.

That principle allows AI systems to reuse meaning, context, computation, and representation while retaining the ability to answer the most important question:

What exact thing did this result come from?

Architecture illustration briefs

- **P05-F01 — The GlyphCAS identity stack**: Stack exact ContentId, logical identity, SemanticId, ContextualReuseKey, DependencyId, RepresentationId, and ReceiptId with arrows showing permissible relationships.
- **P05-F02 — Two linked DAGs**: Exact content Merkle DAG below; semantic object DAG above; each semantic node points to exact supporting chunks.
- **P05-F03 — Layered early exits**: Decision ladder from exact hit through semantic alias, PromptCapsule/prefix/KV reuse, recipe regeneration, and full inference, with cost and verification increasing.
- **P05-F04 — Equivalence modes**: Three panels EXACT_CONTENT, EXACT_SEMANTIC, APPROXIMATE_VERIFIED, emphasizing labels, contracts, and prohibition on promotion to exact identity.

Source register

- **P05-S01** — GlyphCAS Design Report: How We Can Make CAS Lightning Fast for Any LLM (2026-07-24). `How_we_can_make_CAS_lightning_fast_for_any_LLM_by_design_or_adaptation.do`
- **P05-S02** — GlyphDrive/FOVEA/OOC Technical Assessment (2026-07-12). `GLYPHDRIVE-FOVEA-OOC-TECHNICAL-ASSESSMENT-v0.2-PLANNING.md`
- **P05-S03** — GlyphDrive OOC fold/2 Specification v0.2 (2026-07-12). `GLYPHDRIVE-OOC-FOLD-SPEC-v0.2.md`
- **P05-S04** — Zeke 4B Max-Flex Runtime Master Spec (2026-07-15). `ZEKE-4B-MAXFLEX-RUNTIME-MASTER-SPEC-v0.1.md`
- **P05-S05** — Glyph Tokens Main Build Thread Handoff (2026-07-19). `GLYPH_TOKENS_MAIN_BUILD_THREAD_`

Public-disclosure and IP boundary

This edition publishes the public-safe identity architecture, reference evidence, benchmark contract, and threat model. It excludes private compiler prompts, unpublished canonicalization heuristics, customer data, claim-ready patent drafting, and restricted adapter internals. The phrase “lightning fast for any LLM” is treated as a research target, not a public result.