Evidence-Grounded Insurance Advisory Using LandingAI, Amazon Neptune, Redis Vector Search, Centrality-Aware GraphRAG, and Weighted Reciprocal Rank Fusion

Rajesh Kumar Gupta
Rajesh21gupta@gmail.com

July 2026

Executive summary. This paper describes an evidence-grounded insurance advisory architecture that combines document AI, vector retrieval, knowledge graph retrieval, graph centrality, Text2SPARQL, and weighted Reciprocal Rank Fusion (RRF). The objective is to answer insurance-policy questions with high accuracy while preserving traceable evidence down to the source page, chunk, and bounding box. The architecture uses centrality-aware graph ranking, intent-specific scoring, weighted Redis/Neptune fusion, centrality-only graph refresh, and strict evidence-quality controls.

Abstract

Insurance-policy analysis is a high-stakes document intelligence problem. Accurate answers often require exact policy language, state-specific conditions, endorsements, exclusions, limits, and evidence that a reviewer can inspect. A generic language-model response is not sufficient because it may retrieve semantically similar but legally incorrect text, confuse similar carrier language, or produce unsupported citations.

The solution described here implements a hybrid GraphRAG pattern. LandingAI parses policy PDFs into normalized chunks with page and bounding-box metadata. The same normalized chunks are indexed in Redis for vector retrieval and represented in Amazon Neptune as RDF evidence spans and ontology-backed insurance facts. A domain agent retrieves from both stores, ranks graph evidence with quality, ontology, text-match, and centrality signals, applies weighted RRF to combine Redis and Neptune evidence lists, and sends a compact evidence set to the language model. The final answer cites only metadata-backed evidence labels such as `[Evidence 1]`. The user interface resolves those labels into trusted evidence links that open highlighted source images or the original PDF at runtime.

The implementation pattern is applicable to insurance advisory, legal review, compliance analysis, public-sector policy interpretation, contract intelligence, and other regulated workflows where answers must be accurate, explainable, and auditable.

Contents

1 Problem Statement

Insurance documents contain dense contract language, definitions, exclusions, endorsements, limits, duties, notices, and jurisdiction-specific requirements. A correct answer may require several retrieval capabilities at once:

- Semantic recall over policy text.
- Structured lookup of ontology concepts such as coverage, exclusion, endorsement, and limit.
- State or jurisdiction filtering.
- Carrier or document-family comparison.
- Evidence-level traceability to page, chunk, and bounding box.

Vector-only RAG can retrieve relevant-looking text but may miss structured relationships. Graph-only retrieval can provide deterministic facts but may miss nuanced policy wording. A reliable advisory system needs both.

Design principle. The answer must be readable for the business user and verifiable for the reviewer. The language model should not invent source URLs, document references, or evidence labels. Evidence links must be created from backend metadata only.

2 Solution Concept

2.1 Core idea

The solution builds a unified advisory layer over two retrieval systems:

- **Redis vector search** retrieves semantically relevant document chunks.
- **Amazon Neptune RDF graph** retrieves structured insurance facts and evidence relationships through SPARQL.

The two result lists are fused with RRF. The fused context is then used by the language model to produce a concise, evidence-grounded answer.

2.2 Runtime profile

The architecture is parameterized by vector index, named graph, ontology, fusion limits, and evidence-rendering policy.

```
VECTOR_INDEX_NAME = "insurance_policy_vector_index"
NEPTUNE_NAMED_GRAPH = "https://example.org/graph/insurance_advisory"
ONTOLOGY_FILE = "INSURANCE_V2_ONTOLOGY.ttl"
RRF_K = 60
RRF_REDIS_WEIGHT = 1.00
RRF_GRAPH_WEIGHT = 1.12
FUSED_EVIDENCE_INPUT_LIMIT = 50
FUSED_EVIDENCE_LIMIT = 12
FUSED_EVIDENCE_TEXT_LIMIT = 900
EVIDENCE_CITATION_STYLE = "[Evidence n]"
```

2.3 Implementation overview

The retrieval layer uses a staged ranking pipeline:

1. Normalize Redis vector results and Neptune graph rows into one evidence contract.

2. Re-rank graph rows with evidence quality, ontology match, query text match, and graph centrality.
3. Fuse Redis and graph evidence with weighted RRF.
4. Deduplicate by stable chunk identity before citation labels are assigned.
5. Send a compact evidence digest to the synthesis model and prevent the model from inventing evidence URLs.

Practical effect. Redis remains strong at semantic recall. Neptune now contributes structured evidence with a modest graph boost and centrality-aware ordering, so graph facts are less likely to be buried behind generic semantically similar chunks.

2.4 Architecture

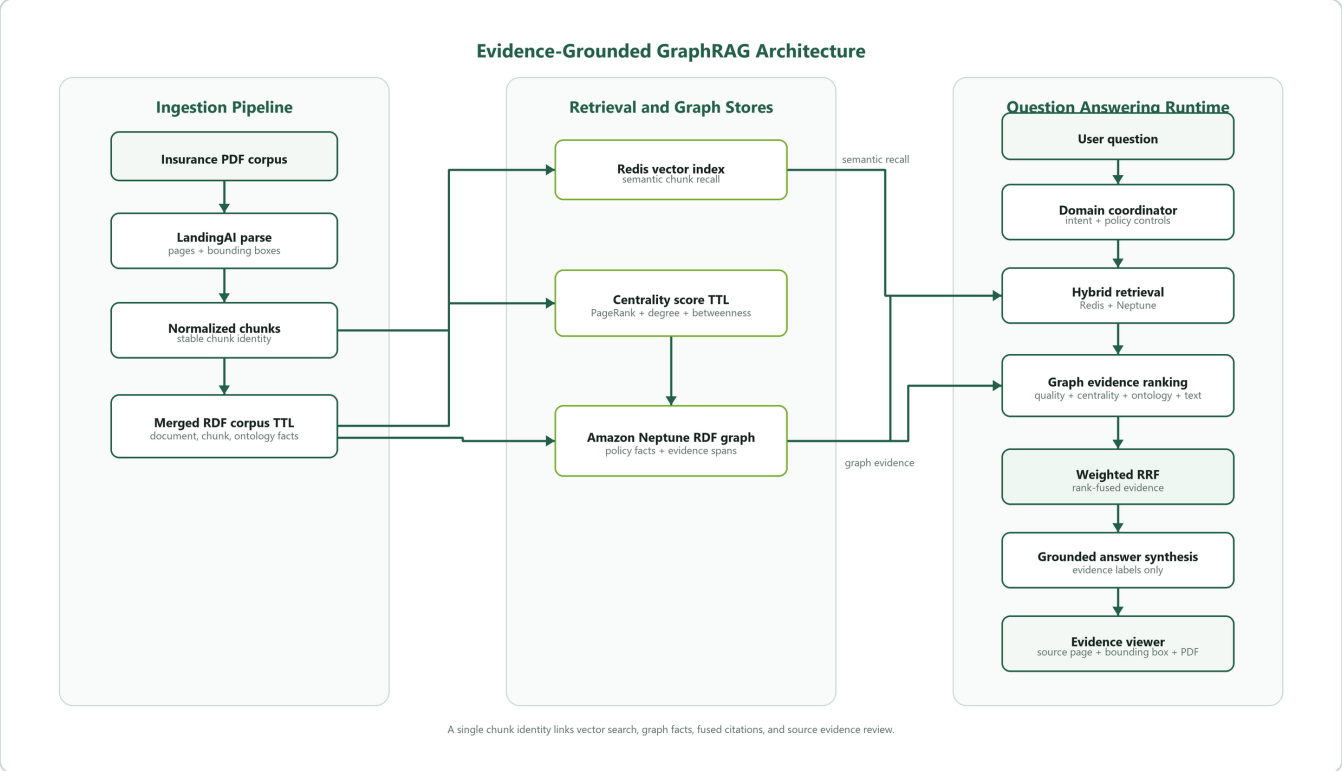

Figure 1: GraphRAG architecture for evidence-grounded insurance advisory.

3 Data and Graph Creation

3.1 Single chunk source

The same normalized document chunks are used for both retrieval stores. This avoids evidence drift between vector search and graph search.

```
1. Parse source PDF with document AI.
2. Create normalized chunks with:
   - chunk_id
   - document_id
   - text
   - markdown
```

```
   - page_start
   - page_end
   - bbox_json
   - source_uri
   - document_category
   - jurisdiction
3. Generate embeddings for the normalized chunks.
4. Store chunks with embeddings in Redis.
5. Store the same chunk identities as RDF evidence spans in Neptune.
```

3.2 Graph construction

The graph is ontology-backed. The TTL generation process creates two classes of RDF content:

- **Evidence nodes**: document, chunk, page, bounding box, snippet, source metadata.
- **Domain fact nodes**: policy, coverage, exclusion, endorsement, limit, deductible, condition, duty, definition, territory, and related concepts.

Each extracted domain fact links back to evidence. This is what enables a graph answer to remain auditable.

```
Source PDF
  -> document_ai_parse_output
  -> normalized_chunks
  -> per_document_ttl
  -> merged_corpus_ttl
  -> Neptune named graph
```

3.3 Ontology coverage

Ontology area	Purpose
Policy entities	Represent policy name, policy type, jurisdiction, form number, and revision metadata.
Coverage entities	Represent coverage names, coverage status, covered parties, covered vehicles, and triggering language.
Exclusion entities	Represent exclusion text, excluded insureds, excluded vehicles, excluded usage, and related carve-outs.
Endorsement entities	Represent endorsement name, form number, revision date, and effect text.
Limit and deductible entities	Represent limits, sublimits, deductible values, basis, and currency.
Evidence entities	Represent chunks, pages, snippets, source documents, bounding boxes, and runtime evidence metadata.

4 Centrality-Aware Graph Layer

4.1 Why centrality was added

Graph retrieval can return many policy facts that are technically connected to the query. Some are central to the policy meaning; others are generic document metadata, cover pages, notices, or boilerplate. Centrality adds a structural prior so the graph can prefer evidence nodes that sit in important policy neighborhoods.

Ranking intent. Centrality does not replace evidence quality or semantic matching. It provides an additional signal that helps Neptune rank structurally important evidence before RRF fusion.

4.2 Implemented centrality algorithms

The implementation supports seven centrality metrics over the merged RDF instance graph. Vocabulary declarations and literal-only edges are excluded from the projection so the score reflects document and policy evidence relationships rather than ontology boilerplate.

Metric	Purpose	Use in policy retrieval
PageRank	Identifies globally authoritative nodes based on incoming graph importance.	Promotes chunks and facts referenced by many important policy relationships.
Degree centrality	Measures local connectedness.	Helps surface evidence connected to several policy concepts, documents, or extracted facts.
Betweenness centrality	Finds bridge nodes between graph neighborhoods.	Useful for diagnosis questions where endorsements, exclusions, and conditions connect otherwise separate policy areas.
Closeness centrality	Measures how near a node is to the rest of the graph.	Useful for broad interpretation when the graph is small enough to compute exactly.
Eigenvector centrality	Rewards nodes connected to other influential nodes.	Helps identify evidence in important policy communities.
Katz centrality	Measures attenuated influence through direct and indirect links.	Useful when influence should decay across longer graph paths.
Harmonic centrality	Handles disconnected graphs better than ordinary closeness.	Useful for corpus graphs where carriers, forms, or jurisdictions form separate components.

4.3 Composite centrality

Each metric is normalized to the range $[0, 1]$. The composite score for node v is:

$$\text{CentralityComposite}(v) = \frac{\sum_{m \in M_{\text{active}}} \alpha_m \cdot \text{norm}_m(v)}{\sum_{m \in M_{\text{active}}} \alpha_m} \tag{1}$$

The default weights are:

```
pagerank     = 0.28
degree       = 0.16
betweenness  = 0.16
closeness    = 0.12
eigenvector  = 0.10
katz         = 0.06
harmonic     = 0.12
```

For large graphs, expensive metrics are guarded by configurable node-count thresholds. If a metric is skipped, the composite is automatically reweighted over the active metrics only. This keeps the refresh practical while preserving a consistent score contract.

4.4 Large-graph behavior

In a representative graph refresh, the merged LandingAI RDF graph projected to 33,118 nodes and 72,156 edges. PageRank, degree centrality, and approximate betweenness centrality were computed. Closeness, eigenvector, Katz, and harmonic centrality were skipped by the large-graph guard because the graph exceeded the configured 4,000-node threshold for those metrics.

```
projection_node_count = 33118
projection_edge_count = 72156
centrality_ttl_triple_count = 132472

active_metrics:
  pagerank
  degree
  betweenness

active_composite_weights:
  pagerank    = 0.466667
  degree      = 0.266667
  betweenness = 0.266667
```

4.5 Centrality RDF contract

Centrality scores are written back as RDF properties on existing graph subjects. This avoids changing chunk identity, document identity, or existing evidence links.

```
ev:centralityPageRank
ev:centralityDegree
ev:centralityBetweenness
ev:centralityCloseness
ev:centralityEigenvector
ev:centralityKatz
ev:centralityHarmonic
ev:centralityComposite
```

When the centrality TTL is loaded into Neptune, the loader first removes the old centrality predicates and then inserts the refreshed score triples. That makes centrality refresh idempotent and avoids duplicate or stale ranking signals.

5 Agent Design

5.1 Logical agent levels

Level	Logical role	Responsibility
L1	Orchestrator	Receives the user request and routes it to the appropriate domain advisory flow.
L2	Domain coordinator	Handles intent, clarity, retrieval orchestration, RRF fusion, prompt construction, answer format, and evidence policy.
L3	Vector retrieval executor	Performs Redis vector retrieval over the configured index.
L3	Graph query executor	Executes SPARQL against the configured Neptune named graph.

L3	Text2SPARQL executor	Converts natural-language graph questions into SPARQL using ontology context.

5.2 Why the execution agents are reusable

The L3 agents are reusable infrastructure components. They do not need to be recreated for every domain because the L2 agent supplies the runtime profile.

```
Domain agent supplies:
  vector_index_name
  neptune_named_graph
  ontology_schema
  retrieval_limits
  evidence_policy
  answer_format
  domain-specific prompt controls
```

This separation allows one platform to support multiple domains while keeping the insurance-specific logic in the domain coordinator.

6 RRF Implementation

6.1 Why RRF is used

Redis and Neptune produce different result types and different score scales.

- Vector search may return cosine distance or similarity.
- Graph retrieval may return SPARQL rows, paths, facts, or rule-ranked results.

These scores are not directly comparable. RRF combines rank positions instead of raw scores.

6.2 Graph pre-ranking

Before fusion, Neptune rows are normalized into evidence items and re-ranked with a graph priority score. The score combines:

- **Evidence quality**: confidence, page, bounding box, snippet, and stable chunk metadata.
- **Centrality**: intent-specific blend of the available centrality fields.
- **Ontology match**: whether the row contains domain concepts such as coverage, exclusion, definition, endorsement, limit, deductible, condition, or duty.
- **Text match**: overlap between meaningful query terms and the evidence snippet.

$$\text{GraphPriority}(e) = w_q Q(e) + w_c C(e) + w_o O(e) + w_t T(e) \tag{2}$$

The weights are selected by inquiry type:

Inquiry	Quality	Centrality	Ontology	Text match
Default	0.40	0.22	0.20	0.18
Interpret	0.36	0.24	0.22	0.18
Compare	0.38	0.18	0.22	0.22
Quantify	0.44	0.10	0.18	0.28
Diagnose	0.32	0.26	0.24	0.18

Communicate	0.42	0.14	0.20	0.24

This makes ranking adaptive. For example, quantify questions lean more heavily on text and evidence quality because exact numbers matter, while diagnose questions give more weight to central bridge nodes and ontology structure.

6.3 Weighted RRF formula

For an evidence item d, the fused score is:

$$\text{WeightedRRF}(d) = \sum_{s \in S(d)} \frac{\lambda_s}{k + r_s(d)} \tag{3}$$

where $r_s(d)$ is the rank of item d in source s, λ_s is the source weight, and k is a dampening constant. The implementation uses $k = 60$, Redis weight 1.00, and graph weight 1.12.

Why graph gets a modest boost. The graph path carries structured evidence relationships and centrality signals. The weight is intentionally small: it helps relevant Neptune evidence compete, but it does not let graph rows dominate if Redis strongly ranks better textual evidence.

6.4 RRF pseudocode

```
inputs:
  redis_results  = ranked evidence from vector search
  graph_results  = Neptune evidence re-ranked by graph priority
  k              = 60
  source_weights = { redis: 1.00, graph: 1.12 }

process:
  score_by_evidence_id = {}

  for each ranked_list in [redis_results, graph_results]:
      for rank, evidence in enumerate(ranked_list, start=1):
          evidence_id = stable_identity(evidence)
          source = evidence.retrieval_source
          score_by_evidence_id[evidence_id] +=
              source_weights[source] / (k + rank)

  fused = sort_by_score_desc(score_by_evidence_id)
  fused = deduplicate_and_attach_metadata(fused)
  fused = take_top_n(fused, FUSED_EVIDENCE_LIMIT)

output:
  fused evidence with page, bbox, snippet, document, and source metadata
```

6.5 Stable evidence identity

Deduplication uses the most stable available identifier:

```
1. landingai_chunk_id
2. source_element_id
3. chunk_id
4. fallback hash(document_id, file_name, page, snippet)
```

This prevents the same evidence from appearing twice when Redis and Neptune return it through different paths. If both sources return the same evidence, the final source label becomes `hybrid`.

6.6 Retrieval flow

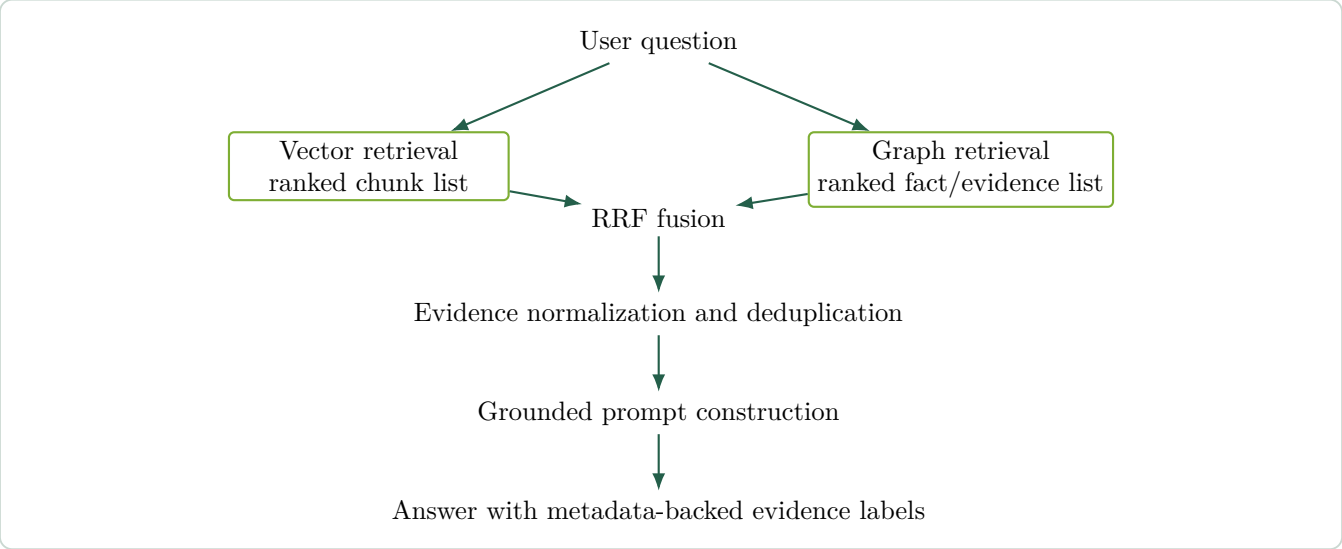

Figure 2: RRF combines vector and graph retrieval before answer synthesis.

7 Evidence Strategy

7.1 Clean answer citations

The answer format keeps the text clean. Instead of printing long evidence URLs inline, the answer cites evidence labels.

```
Example answer sentence:

- Carrier A excludes damage for the described scenario when the policy
  language matches the stated exclusion conditions [Evidence 1].
- Carrier B uses different endorsement language and should be reviewed
  separately for the same scenario [Evidence 2].
```

7.2 Evidence-link rule

Strict rule. The language model must not create evidence URLs. It may only reference evidence labels that are present in backend metadata. The user interface creates the clickable link from trusted metadata.

7.3 Runtime evidence rendering

The evidence viewer uses document metadata to reconstruct the source page image at runtime and overlay the bounding box. The user-facing view should prioritize review actions, not raw metadata.

- **Open original PDF**: a clear button that resolves the source S3 URI or a signed URL when available.
- **Highlighted source image**: the exact page region with the LandingAI bounding box overlay.
- **Minimal citation context**: document name, page number, and short snippet.

- **Debug metadata only when needed**: document id, chunk id, bounding box JSON, source element id, and evidence JSON should be hidden from ordinary users or placed behind a debug affordance.

7.4 Citation consistency checks

The verification pass on comparative policy questions showed that a mathematically relevant evidence item can still be semantically misapplied. For production, the answer layer should enforce claim-citation consistency:

Check	Purpose
Carrier consistency	A Chubb claim should not cite PURE evidence, and a PURE claim should not cite Chubb evidence.
Document-category consistency	A valuation claim should cite valuation or limit-of-liability language, not a cover page or rating disclosure.
Amendment precedence	If a state amendment replaces base policy language, the answer should mention the amendment or scope the answer to the base form.
Material-claim support	Every recommendation or comparison conclusion should point to evidence that actually supports the claim.
Citation compactness	Inline evidence must render as compact labels or pills, not long raw URLs inside tables.

8 SPARQL Capability

The graph supports both fixed templates and dynamic Text2SPARQL. Fixed templates are useful for common advisory workflows. Text2SPARQL handles exploratory graph questions.

Query family	Supported use
Interpret	Definitions, exclusions, coverages, endorsements, notices, procedures.
Compare	Carrier or document-family comparison across definitions, coverages, exclusions, and endorsements.
Quantify	Limits, sublimits, deductibles, and rating factors.
Diagnose	Conditions, coverage status, underwriting constraints, and exclusion applicability.
Communicate	Policy summaries, key exclusions, coverages, declarations, and evidence-backed explanations.
Probe	Graph validation, document counts, policy type counts, carrier counts, and ontology coverage checks.

8.1 Centrality fields in SPARQL results

SPARQL templates now return centrality fields when they are available on chunks, facts, exclusions, duties, or related evidence nodes. The coordinator consumes these fields during graph pre-ranking.

```
SELECT
  ?chunk
  ?snippet
  ?documentId
  ?fileName
  ?page
  ?bbox
  ?centralityPageRank
  ?centralityDegree
  ?centralityBetweenness
  ?centralityCloseness
```

```
  ?centralityEigenvector
  ?centralityKatz
  ?centralityHarmonic
  ?centralityComposite
WHERE {
  ?chunk ev:snippet ?snippet .
  OPTIONAL { ?chunk ev:centralityPageRank ?centralityPageRank . }
  OPTIONAL { ?chunk ev:centralityDegree ?centralityDegree . }
  OPTIONAL { ?chunk ev:centralityBetweenness ?centralityBetweenness . }
  OPTIONAL { ?chunk ev:centralityComposite ?centralityComposite . }
}
```

The real templates use `COALESCE` across possible node locations so centrality can be read from the evidence chunk or from the related domain fact node. This keeps ranking useful even when different query families return different graph shapes.

8.2 Example SPARQL query

All code is boxed to prevent page spill and improve readability.

```
PREFIX ins:  <https://example.org/ontology/insurance#>
PREFIX ev:   <https://example.org/ontology/evidence#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT
  ?endorsement
  ?name
  ?formNumber
  ?formRevisionDate
  ?documentId
  ?fileName
WHERE {
  GRAPH <https://example.org/graph/insurance_advisory> {
    ?endorsement a ins:Endorsement .

    OPTIONAL { ?endorsement ins:endorsementName ?endorsementName . }
    OPTIONAL { ?endorsement rdfs:label ?label . }
    BIND(COALESCE(?endorsementName, ?label, STR(?endorsement)) AS ?name)

    OPTIONAL { ?endorsement ins:formNumber ?formNumber . }
    OPTIONAL { ?endorsement ins:formRevisionDate ?formRevisionDate . }

    OPTIONAL {
      ?endorsement ev:sourceDocument ?sourceDocument .
      OPTIONAL { ?sourceDocument ev:documentId ?documentId . }
      OPTIONAL { ?sourceDocument ev:fileName ?fileName . }
    }
  }
}
ORDER BY ?name ?formNumber
```

9 Accuracy Evaluation

9.1 Evaluation dimensions

Metric	Meaning
Answer correctness	The answer matches the source policy language and graph facts.
Evidence support rate	Every material claim has a supporting evidence label.

Citation precision	The linked evidence actually supports the cited claim.
Source consistency	Carrier, document family, jurisdiction, and policy type match the claim being cited.
Retrieval recall	The fused evidence set includes the necessary source chunks or graph facts.
Graph fact accuracy	Extracted ontology facts correctly represent the source document.
No-hallucination rate	The answer does not invent policy details, evidence labels, or URLs.
Latency	The answer returns within an acceptable advisory workflow time.
Comparison quality	Carrier or document comparisons are fair, scoped, and clearly separated.

9.2 Retrieval quality findings

Verification of comparative policy questions highlights the value of centrality-aware fusion and the importance of citation-quality gates:

- Graph evidence is now participating in fused results instead of Redis-only retrieval.
- The high-level valuation formulas can be retrieved correctly when the right Chubb and PURE limit-of-liability chunks appear.
- A citation can still be wrong if a Chubb claim cites PURE evidence or if an answer ignores an applicable state amendment.
- Generic pages such as cover pages, rating notices, or mileage disclosures should be pushed down unless the query asks for them.
- Table rendering must keep evidence labels compact so citations do not overlap answer text.

These findings turn into concrete production tests: carrier-citation alignment, amendment precedence, citation relevance, and UI overflow checks.

9.3 Golden-question coverage

The test set should include:

- coverage questions;
- exclusion questions;
- endorsement questions;
- limit and deductible questions;
- jurisdiction-scoped questions;
- carrier or document-family comparison questions;
- ambiguous questions requiring clarification;
- out-of-scope questions requiring refusal.

9.4 Representative test questions

1. How do two carriers differ in coverage for damage to a covered auto?
2. List all exclusions for a state-specific policy document.
3. List all endorsements present in the knowledge graph.
4. What does a supplemental liability endorsement do?
5. Which exclusion is modified by a given endorsement?

6. What does a glass coverage endorsement cover?
7. Compare owned-vehicle exclusions across two policy families.
8. What are the duties after an accident or loss?
9. Which evidence supports the answer about no-fault coverage?
10. Show the source evidence for the top exclusion result.

10 Deployment and Refresh Strategy

10.1 Do not rerun ADE when the PDFs have not changed

The centrality and ranking updates are graph-side and coordinator-side improvements. If the source PDFs, LandingAI parse outputs, normalized chunks, embeddings, and extracted RDF facts have not changed, there is no need to rerun the expensive document AI path.

Centrality refresh path. The existing merged corpus TTL is sufficient for computing centrality. Loading only the centrality predicates into Neptune refreshes graph ranking without reprocessing PDFs through ADE.

10.2 Centrality-only refresh

```
Inputs:
  existing merged_corpus.ttl
  existing Redis vector index
  existing Neptune named graph

Steps:
  1. Build graph projection from merged_corpus.ttl.
  2. Compute centrality metrics with large-graph guards.
  3. Write graph_centrality_scores.json.
  4. Write graph_centrality.ttl.
  5. Delete old centrality predicates in Neptune.
  6. Insert refreshed centrality triples.
  7. Restart or redeploy the coordinator service.

No ADE/PDF parse is required.
No chunk regeneration is required.
No Redis embedding rebuild is required.
```

10.3 When full ingestion is required

Full ingestion should be rerun only when upstream document or extraction artifacts change.

Change	Required action
New PDF or revised PDF	Rerun ADE parse, chunking, embedding, TTL generation, Redis indexing, and Neptune load.
Changed chunking rules	Rerun chunk generation and downstream Redis/Neptune loads because chunk identities may change.
Changed ontology extraction logic	Regenerate RDF facts and reload graph artifacts.
Changed embedding model or vector schema	Rebuild Redis embeddings and index.
Changed centrality weights or algorithms only	Recompute centrality TTL and reload centrality predicates only.

Changed RRF/source weights only	Deploy coordinator configuration or code only; no ingestion required.
Changed UI evidence rendering only	Deploy frontend only; no ingestion required.

10.4 Operational safety

The Neptune loader writes a load marker with TTL hash, artifact kind, load status, timestamp, batch count, and graph URI. Centrality loads use replaceable predicates so the graph can be refreshed repeatedly without accumulating stale score triples.

11 Production Readiness

11.1 Controls

Control area	Implementation direction
Graph isolation	Use a dedicated Neptune named graph per corpus or environment.
Vector isolation	Use a dedicated Redis index per corpus or environment.
Evidence integrity	Generate evidence links from backend metadata only.
Data lineage	Preserve source document id, chunk id, page, bounding box, and source URI.
Access control	Protect the UI, API, document store, Redis, and Neptune endpoints.
Secrets	Store keys and endpoints in managed secrets or parameter stores.
Observability	Log retrieval counts, RRF inputs, fused evidence count, latency, and failures.
Evaluation	Run golden-question accuracy checks before release.
Fallback	Keep a deterministic no-evidence response for insufficient data.

11.2 Deployment checklist

```
Production release checklist:

[ ] Confirm all source documents are approved for the intended audience.
[ ] Redact carrier names, internal namespaces, and private graph URIs if needed.
[ ] Verify Redis index and Neptune graph point to the intended corpus.
[ ] Run golden-question evaluation and save the result log.
[ ] Verify evidence links open source images with correct bounding boxes.
[ ] Confirm the model cannot invent evidence URLs.
[ ] Confirm authentication and authorization are enabled.
[ ] Confirm secrets are not stored in source code.
[ ] Capture a short offline demo in case live services are unavailable.
```

12 Business Value

The architecture creates value in any document-heavy advisory workflow:

- **Higher trust:** every answer can be traced to source evidence.
- **Better recall:** vector retrieval captures semantic matches.
- **Better precision:** graph retrieval captures structured policy relationships.
- **Lower hallucination risk:** the answer is constrained to fused evidence.

- **Reusable platform:** the same L3 execution agents can support multiple domain agents.
- **Auditability:** evidence metadata supports compliance and quality review.

13 Conclusion

This solution demonstrates a practical pattern for high-accuracy, evidence-grounded advisory AI. The main innovation is the combination of:

- document AI evidence extraction;
- shared chunk identity across Redis and Neptune;
- ontology-backed RDF graph modeling;
- centrality-aware graph ranking over the merged evidence graph;
- reusable graph and vector execution agents;
- weighted RRF rank fusion;
- metadata-backed evidence links in the user interface.

The result is a GraphRAG system that can answer complex insurance-policy questions while giving reviewers direct access to the evidence behind each claim.

A Neutral One-Page Summary

Title: Evidence-Grounded GraphRAG for Insurance Advisory
Problem: Insurance-policy questions require exact language, structured policy concepts, and reviewable evidence. Generic summarization is not reliable enough for regulated workflows.
Solution: Parse documents into grounded chunks, store the same chunk identities in Redis and Neptune, model insurance concepts with an ontology, compute graph centrality over the merged evidence graph, retrieve from both vector and graph stores, re-rank graph evidence with centrality and quality signals, fuse ranked evidence with weighted RRF, and produce a concise answer with metadata-backed evidence labels.
Outcome: Business users receive clean answers. Reviewers can click evidence labels to inspect source pages and highlighted bounding boxes. The architecture reduces hallucination risk and creates a reusable pattern for document-heavy advisory domains.

B Reference Architecture Profile

```
Layer 1: Orchestrator
  - Receives user query
  - Routes to domain advisory flow

Layer 2: Domain coordinator
  - Classifies intent
  - Calls vector retrieval
  - Calls graph retrieval
  - Calls Text2SPARQL when needed
  - Applies centrality-aware graph re-ranking
  - Applies weighted RRF
  - Builds grounded prompt
  - Returns answer plus evidence metadata

Layer 3: Execution agents
  - Redis vector retrieval executor
  - Neptune SPARQL executor
  - Text2SPARQL executor

User interface
  - Renders answer
  - Converts [Evidence n] labels into clickable links
  - Opens runtime evidence viewer
  - Provides original PDF access when a trusted source URI exists
```

C References

References

[1] Amazon Neptune product documentation. https://aws.amazon.com/neptune/

[2] Amazon Neptune documentation: accessing a graph with SPARQL. https://docs.aws.amazon.com/neptune/latest/userguide/access-graph-sparql.html

[3] AWS Database Blog: model-driven graphs using OWL in Amazon Neptune. https://aws.amazon.com/blogs/database/model-driven-graphs-using-owl-in-amazon-neptune/

[4] LandingAI documentation. https://docs.landing.ai/

[5] Original publication on Reciprocal Rank Fusion. https://research.google/pubs/reciprocal-rank-fusion-outperforms-condorcet-and-individual-rank-learning-methods/

[6] Original PageRank research paper. https://research.google/pubs/the-pagerank-citation-ranking-bringing-order-to-the-web/

[7] NetworkX graph algorithms documentation. https://networkx.org/documentation/stable/reference/algorithms/centrality.html