

Distributed Passive Retrieval (DPR) with Resonant Consensus

System Architecture & Reference Implementation

Alan J Garcia

December 13, 2025

Abstract

Despite substantial investments in training Large Language Models on vast datasets, industry standards for long-term memory remain surprisingly primitive. Most systems today rely on flat retrieval mechanisms that pull text based on keyword matching, devoid of deeper context or verification. This document outlines the **Distributed Passive Retrieval with Resonant Consensus (DPR-RC)** architecture. Instead of a static database, the system employs a fleet of "Passive Agents" that retain, verify, and vote on their knowledge. It is a system designed to deliver verified consensus, prioritizing accuracy over the immediacy of hallucination.

1 Executive Summary

Consider the limitations of first-generation search engines. They operated on a simple principle: if the keyword appeared in the text, it was a match. This flat, text-level approach mirrors the current state of Retrieval Augmented Generation (RAG). One feeds an embedding model text, and whatever vectors align are injected into the prompt. The depth of analysis effectively ends there.

DPR-RC advances beyond flat text by establishing a network of historical agents that function as a coordinated research team.

1. **Time-Sharded Routing (L1):** The system narrows the retrieval field to specific, relevant historical eras.
2. **Distributed Verification (L2):** Small Language Models (SLMs) are employed to reason about the data, rather than merely matching keywords.
3. **Resonant Consensus (L3):** Agents peer-review each other's findings to filter out noise and mitigate hallucinations.

Strategic Note: This architecture is not designed for sub-millisecond response times. We accept higher latency (2–10s) to achieve verifiable, high-precision recall. For speed, a cache is sufficient; for verifiable historical truth, DPR is required.

2 System Concept

The core philosophy is: "**Index the Agent, Verify the Context, Synthesize the Consensus.**"

Opinions and memories exist within a network of relations, not in a vacuum. Traditional systems obscure these connections by fragmenting documents into disconnected chunks. DPR addresses this by interacting with a sequence of historical agents. When the **Active Agent (A^*)** requires information, it does not merely query a database. It identifies past versions of itself, activates them, and initiates a consensus protocol to validate the retrieved information.

2.1 Key Definitions

Active Agent (A^*) The live interface managing the immediate context window.

Passive Agent (\mathcal{P}) A serverless worker representing a frozen snapshot of history, performing verification on demand.

Foveated Search The mechanism focusing high-compute resources on specific, relevant points in history while ignoring irrelevant data.

Resonant Consensus A protocol where passive agents peer-review retrieved memories, mapping results to a *Semantic Quadrant Topology* to distinguish agreed-upon facts from perspectival differences.

3 High-Level Architecture

The system is composed of four decoupled components deployed on cloud infrastructure (e.g., Google Cloud Platform).

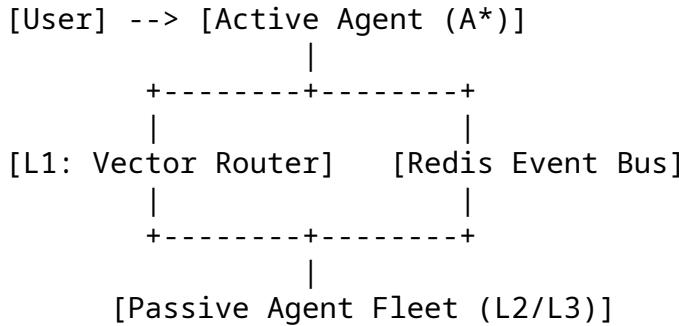


Figure 1: High-Level Data Flow: Routing → Verification → Consensus

3.1 Component Breakdown

3.1.1 1. The Time-Sharded Vector Index (L1)

To ensure scalability, the Central Index is partitioned into **Time-Based Shards** (e.g., "Recent", "Archive").

- **Role:** Rapid filtering via parallel queries.
- **Mechanism:** Approximate Nearest Neighbor (ANN) search.
- **Output:** A targeted list of Agent IDs.

3.1.2 2. The Event Bus (Redis Streams)

A high-throughput message broker that decouples the Active Agent from the Passive Fleet, preventing synchronous blocking.

3.1.3 3. The Passive Agent Fleet (L2/L3)

Serverless workers representing specific epochs of history.

- **Mechanism:** Uses a quantized SLM (e.g., Phi-3) to reason about the query against frozen context.
- **Benefit:** Performs reasoning ("Does this context answer the specific question?") rather than simple similarity matching.
- **Deployment:** utilizes "Scale-to-Zero" infrastructure; cold start latency is an accepted trade-off for infinite memory at minimal resting cost.

4 The DPR Protocol

The system operates in a continuous loop defined by the **RFI (Request for Information)** protocol.

4.1 Step 1: Gap Detection & Routing

A^* determines it needs external information.

1. A^* embeds query Q into a vector.
2. A^* queries L1 Indices to find the Top- K (e.g., 5–50) relevant agents.

4.2 Step 2: Targeted Broadcast

A^* publishes an RFI message to Redis containing a **Routing Key**.

- Payload: { "query": Q , "target_agents": ["agent-id-1", ...] }

4.3 Step 3: Distributed Verification (L2)

Targeted Passive Agents wake up and ingest the query.

1. Load local frozen context.
2. Run **SLM** verification: "Does context contain answer to Q ?"
3. Calculate Confidence Score C based on SLM probability and memory depth.

4.4 Step 4: Resonant Consensus (L3)

Instead of returning the single best answer, the system enters a Consensus Phase.

1. **Peer Review:** Agents broadcast findings to a temporary voting channel.
2. **Evaluation:** Agents vote on the validity of others' findings based on their own context.
3. **Topology Mapping:** A^* maps results to a Semantic Quadrant Topology to determine consensus or conflict.

4.5 Step 5: Superposition Injection

A^* injects a **Superposition Object** into its context, containing both the **Consensus Truth** and distinct **Perspectives**.

5 Reference Implementation (Python)

5.1 Active Agent (Controller)

Handles L1 Routing and RFI Broadcast.

```
1 import redis
2 import json
3 import uuid
4 import chromadb
5 from sentence_transformers import SentenceTransformer
6
7 class ActiveAgentController:
8     def __init__(self, redis_host, chroma_paths):
9         self.bus = redis.Redis(host=redis_host, decode_responses=True)
10        # Initialize Sharded Indices
11        self.l1_shards = {
12            name: chromadb.PersistentClient(path=path).get_collection("agent_summaries")
13            for name, path in chroma_paths.items()
14        }
15        self.embedder = SentenceTransformer('all-MiniLM-L6-v2')
16
17    def ask_history(self, user_query, shards_to_query=['recent', 'archive']):
18        query_vec = self.embedder.encode(user_query).tolist()
19        all_target_ids = []
20
21        # 1. Query Time Shards
22        for shard_name in shards_to_query:
23            if shard_name in self.l1_shards:
24                results = self.l1_shards[shard_name].query(
25                    query_embeddings=[query_vec], n_results=3
26                )
27                if results['ids'][0]:
28                    all_target_ids.extend(results['ids'][0])
29
30        if not all_target_ids: return None
31
32        # 2. Broadcast Targeted RFI
33        payload = {
34            "rfi_id": str(uuid.uuid4()),
35            "query": user_query,
36            "target_agents": json.dumps(all_target_ids)
37        }
38        self.bus.xadd("stream:rfi", payload)
39        return payload['rfi_id']
```

5.2 Passive Agent (Worker)

Performs L2 Verification using an SLM.

```
1 import json
2 import redis
3 from llm_engine import SLMRunner
4
5 class PassiveAgentWorker:
6     def __init__(self, agent_id, frozen_context, redis_host):
```

```

7     self.agent_id = agent_id
8     self.context = frozen_context
9     self.bus = redis.Redis(host=redis_host, decode_responses=True)
10    # L2 Verification: 4-bit Quantized SLM
11    self.verifier = SLMRunner('microsoft/Phi-3-mini-4k-instruct', quantization='4bit'
12 )
13
14    def process_rfi(self, rfi_payload):
15        # 1. Routing Check
16        targets = json.loads(rfi_payload['target_agents'])
17        if self.agent_id not in targets: return
18
19        # 2. Deep Verification via SLM
20        query = rfi_payload['query']
21        context_str = "\n".join(self.context)
22        prompt = f"Context:\n{context_str}\nQuestion: {query}\nTask: Does the context
23        contain the answer?"
24
25        result = self.verifier.generate(prompt)
26        parsed = json.loads(result)
27
28        # 3. Broadcast to Voting Channel
29        if parsed['has_answer'] and parsed['confidence'] > 0.7:
30            response = {
31                "agent_id": self.agent_id,
32                "content": parsed['extracted_answer'],
33                "confidence": float(parsed['confidence'])}
34            self.bus.xadd("stream:voting", response)

```

6 Simulation Scenarios

This section details two reference simulations demonstrating the Resonant Consensus Protocol in action. These examples illustrate how the system handles converging narratives versus irreconcilable conflicts.

6.1 Scenario A: The "Fog of War" Consensus

Context: Agents from different time points (During Incident, Post-Fix, Post-Mortem) have conflicting immediate observations but converge on a single root cause.

User Query: "Why did the payment API fail on Oct 5th?"

L1 Routing Result:

- Agent_T0 (Fog of War): Active during incident (Oct 5, 2:00 PM).
- Agent_T1 (Immediate Fix): Active post-fix (Oct 5, 4:00 PM).
- Agent_T2 (Retrospective): Active post-mortem (Oct 12).

Step 1: RFI Broadcast

```

1 {
2   "rfi_id": "rfi-temporal-001",
3   "query": "Why did the payment API fail on Oct 5th?",
4   "target_agents": ["Agent_T0", "Agent_T1", "Agent_T2"],
5   "context": "incident_analysis"
6 }
```

Step 2: Initial Findings

- **Proposal A (Agent_T0):** "We are under a DDoS attack. Traffic spiked 500%." (Confidence: 0.95)
- **Proposal B (Agent_T1):** "Not DDoS. Bad deploy (v2.1) caused memory leak. Rolled back." (Confidence: 0.99)
- **Proposal C (Agent_T2):** "Root cause: Recursive retry loop in client library v2.1 caused self-inflicted DDoS." (Confidence: 0.99)

Step 3: Peer Review (Voting)

- **Agent_T2 votes on A:** Negative Asymmetry. "Traffic was internal, not external. Misdiagnosis."
- **Agent_T0 votes on C:** Neutral/Positive. "Explains why WAF blocked requests. Fits observations."

Step 4: Topological Mapping

- **Proposal A (DDoS):** Rejected by future agents → Asymmetric Perspective (Historical Misconception).
- **Proposal C (Retry Loop):** Agreed by future agents, accepted by past agents → Symmetric Resonance (Consensus Truth).

Step 5: Final Response

```

1 {
2   "rfi_id": "rfi-temporal-001",
3   "superposition": {
4     "symmetric_resonance": {
5       "content": "Root cause was recursive retry loop in client library v2.1.",
6       "consensus_score": 0.95,
7       "source": "Agent_T2 (Post-Mortem)"
8     },
9     "asymmetric_perspectives": [
10       {
11         "type": "Historical Context",
12         "content": "Initially misdiagnosed as DDoS attack due to WAF volume.",
13         "source": "Agent_T0 (Incident Log)"
14       }
15     ]
16   }
17 }
```

6.2 Scenario B: The "Phantom Feature" Conflict

Context: A policy change (privacy update) creates disjoint truths where no single answer satisfies all historical contexts.

User Query: "Does the get_user_data API return the email field by default?"

L1 Routing Result:

- Agent_2021 (Legacy): Active when email was public.
- Agent_2023 (Modern): Active after strict PII lockdown.

Step 2: Initial Findings

- **Proposal A (Agent_2021):** "Yes, returns full user object including email." (Confidence: 0.99)
- **Proposal B (Agent_2023):** "No. Email is redacted. Requires special scope." (Confidence: 0.99)

Step 3: Peer Review (Total Polarization)

- **Agent_2023 votes on A:** Strong Negative (-1.0). "False and dangerous. P0 security violation."
- **Agent_2021 votes on B:** Strong Negative (-1.0). "Incorrect. Endpoint doesn't exist."

Step 4: Topological Mapping

Both proposals fall into Asymmetric quadrants. The Consensus quadrant is empty.

Step 5: Final Response (Divergence Report)

```
1 {
2     "rfi_id": "rfi-conflict-005",
3     "status": "consensus_failed",
4     "superposition": {
5         "symmetric_resonance": null,
6         "conflict_analysis": "Irreconcilable divergence between 2021 and 2023 epochs.",
7         "asymmetric_perspectives": [
8             {
9                 "perspective": "Legacy (Pre-2022)",
10                "content": "Yes, returned by default.",
11                "source": "Agent_2021"
12            },
13            {
14                "perspective": "Modern (Post-2022)",
15                "content": "No, redacted for privacy.",
16                "source": "Agent_2023"
17            }
18        ]
19    }
20 }
```

7 Performance & Tradeoff Analysis

Feature	Standard RAG	Native Long-Context	DPR (Proposed)
Search Unit	Document Chunk	Entire History	Historical Agent State
Primary Goal	Latency	Comprehension	Scalable High Fidelity
Latency	Low (< 50ms)	High (5s+)	Asynchronous (2–10s)
Fidelity	Low (Keyword bias)	High (Full Attention)	High (Reasoning-Verified)
Cost at Scale	Low	Very High	Foveated

Table 1: Architectural Comparison

8 Lifecycle Management (Embedding Drift)

Addressing Semantic Drift—the inevitable shift in embedding models over time—requires a strategy beyond simply re-indexing petabytes of data. We employ an **Adapter Strategy**, training a linear adapter W that maps old embeddings to new spaces. This enables a lazy migration strategy that keeps costs down.

9 Caching & Optimization

The system implements a **Reinforced Memory** cache. If an agent returns a super-high confidence response ($\mathcal{C} > 0.9$), that ‘(query, response)’ pair is cached in a “Hot Memory” shard. Subsequent requests skip the L2 verification step, delivering the answer directly.

10 Cost Modeling

Comparing DPR (Serverless) vs. Dedicated GPU for 10,000 daily queries targeting 10-year history:

- **DPR (Cloud Run):** $\approx \$10 - \$15/\text{day}$. Costs scale to zero when idle. Superior for bursty workloads.
- **Dedicated GPU:** $\approx \$12 - \$18/\text{day}$ (Always On). Superior for continuous baseload but lacks elasticity.

11 Evaluation Methodology

Unlike standard benchmarks which prioritize search latency and document recall, DPR-RC requires evaluation on axes of temporal fidelity and conflict resolution. We solve the “no benchmarks” problem by creating synthetic histories where we possess ground truth over the evolution of facts.

11.1 10.1 The “Long-Horizon” Synthetic Benchmark

Standard datasets like MS MARCO do not account for when a fact was true. We generate a deterministic 10-year research history where “truth” is a function of time.

```

1 def generate_research_history(years=10, events_per_year=500):
2     history = []
3     for year in range(years):
4         for event in range(events_per_year):
5             # Deterministically mutate "truth" over time
6             record = {
7                 "timestamp": f"{2015 + year}-{random_month()}-{random_day()}",
8                 "content": generate_research_event(),
9                 "conclusions": generate_conclusions(), # May contradict earlier years
10                "perspective": random.choice(["methodological", "theoretical"])
11            }
12            history.append(record)
13    return history
14
15 def generate_temporal_queries(history):
16     # Test if system retrieves the 2018 truth or the 2020 truth
17     queries = []
18     queries.append({
19         "query": "What was our understanding of X in 2018?",
20         "ground_truth": filter_by_time(history, "2018"),
21         "type": "temporal_recall"
22     })
23     return queries

```

11.2 10.2 The Ablation Protocol

To justify the complexity of the Resonant Consensus layer, we employ an ablation strategy. By toggling $L3 = Off$, we measure the baseline hallucination rate of simple retrieval. If enabling $L3$ reduces hallucinations by 40% while increasing latency by 2s, the engineering tradeoff is quantitatively defensible.

11.3 10.3 The Evaluation Meta-Point

Existing benchmarks measure **Search** (finding text). DPR measures **Synthesis** (finding truth). The fact that DPR scores 'N/A' on standard benchmarks is not a failure; it is proof that we are addressing a problem scope—structured consensus over long-horizon episodic memory—that the industry has not yet standardized.