memory-spark: GPU-Accelerated Persistent Memory for Autonomous AI Agents

A Production System for Agentic Retrieval-Augmented Generation with Dynamic Reranker Gating, Reciprocal Rank Fusion, and Multi-Pool Semantic Memory

Klein Panic[1]

[1]Independent Researcher, klein@kleinpanic.com

April 2026

Abstract

We present **memory-spark**, a production memory substrate for autonomous AI agents that continuously ingests workspace knowledge, indexes it with hybrid dense–sparse retrieval, and injects high-relevance context before each conversational turn. Unlike standard Retrieval-Augmented Generation (RAG) systems designed for single-turn question answering, memory-spark is engineered for the fundamentally harder problem of persistent agentic memory: heterogeneous source types (workspace files, tool schemas, mistake logs, reference documents, captured preferences), strict sub-second latency requirements (memory injected before *every* turn, not just user queries), cross-agent knowledge sharing with data isolation, and active defense against self-poisoning from agent-generated errors.

The system operates entirely on local GPU infrastructure (NVIDIA DGX Spark) with no cloud API dependencies for embeddings, reranking, or LLM inference. We introduce three key technical contributions: (1) a **Dynamic Reranker Gate** that analyzes first-stage vector score distributions to intelligently bypass cross-encoder reranking when retrieval confidence is already high, reducing latency by ∼50% with no quality loss; (2) **scale-invariant Reciprocal Rank Fusion** (RRF) for hybrid dense–sparse merging and retrieve–rerank blending, resolving the fundamental incompatibility between BM25 score ranges (5–20+) and cosine similarity ranges (0.2–0.6) that caused catastrophic rank-washout in prior score-based approaches; and (3) a **15-stage retrieval pipeline** with instruction-aware asymmetric embeddings, multi-pool search, source priority weighting, temporal decay, Hypothetical Document Embeddings (HyDE), multi-query expansion, Maximal Marginal Relevance (MMR) diversity, parent-child chunk expansion, and layered security filtering.

We evaluate on the BEIR benchmark [Thakur et al., 2021] across three datasets. On SciFact (300 queries), our production configuration GATE-A achieves NDCG@10 of 0.7802, Recall@10 of 0.9137, and mean latency of 626ms while invoking the cross-encoder on only 21% of queries. Our best configuration (Config U, logit blend $\alpha = 0.4$) achieves NDCG@10 of 0.7889 and Recall@10 of 0.9099. On FiQA (648 queries, financial Q&A), GATE-A achieves 0.5479 NDCG@10 (+66% over BM25, +68% over Contriever). On NFCorpus (323 queries, cross-domain medical retrieval), Vector-Only achieves 0.4443 NDCG@10 (+35.5% over Contriever), while reranking *hurts* performance due to the cross-domain query format mismatch.

A detailed forensic analysis of the 12-phase development cycle documents eight categories of production failure modes, including silent NaN propagation via Apache Arrow type mismatches, BM25 sigmoid score saturation causing rank-washout, cross-encoder score compression, and

pipeline stage ordering bugs. We argue that the gap between benchmark RAG and production agentic RAG is larger than commonly acknowledged, and that iterative quantitative evaluation (not intuition) is the only reliable guide to system improvement.

The complete system is validated in an isolated Docker harness with 18 self-service memory tools, 37,000+ indexed chunks, and a 483-test validation suite. All code, benchmarks, and evaluation scripts are open-source at https://github.com/kleinpanic/memory-spark.

1 Introduction

1.1 The Persistent Memory Problem for Autonomous Agents

Long-running AI agents face a memory problem that is categorically different from the context management challenges of single-turn assistants. A conversational assistant must remember what was said ten messages ago. An autonomous agent running 24 hours a day, executing dozens of tasks per session across hundreds of sessions, must remember what was learned *weeks* ago—that a particular configuration key must never be modified during live operation, that a previous attempt at a seemingly safe change caused a production outage, that a specific file path is sacred, that a colleague prefers concise responses over detailed reports.

The context window of even the largest LLMs (128k–1M tokens) is insufficient for this task, and for good reason: naïvely injecting everything is both token-expensive and architecturally fragile. A 128k token context filled with agent logs leaves little room for the actual task. More fundamentally, irrelevant context degrades performance [Shi et al., 2023]: a model trying to use 50,000 tokens of historical workspace content to answer a simple question about a configuration value performs worse than one given the 200 most relevant tokens.

Retrieval-Augmented Generation (RAG) [Lewis et al., 2020] addresses this by retrieving relevant documents at query time. But production agentic RAG introduces challenges absent from standard QA benchmarks:

Source heterogeneity. Agent workspaces contain markdown documentation, TypeScript source code, tool schemas, daily memory notes, mistake logs with root cause analyses, captured user preferences, reference PDFs, and session transcripts. Each source type has different retrieval characteristics, different relevance signals, and different weighting requirements. A mistake entry from three months ago that is *exactly* applicable to the current task should outrank a generic document that scores slightly higher on cosine similarity.

Latency constraints. In production, memory retrieval happens before *every* agent turn—not just when the user explicitly asks a factual question. If retrieval takes 3 seconds, every conversational exchange has a 3-second tax. This is unacceptable for interactive use. Sub-second latency is a hard requirement.

Temporal dynamics. Recent information should generally outweigh old information, but not uniformly. A user preference captured yesterday is more reliable than one from six months ago. But a safety rule ("never edit openclaw.json during a heartbeat run") is timeless and should not be temporally discounted.

Cross-agent sharing. A multi-agent system where agents can share knowledge but must maintain data isolation presents a design tension. Agent A's per-session scratchpad should not leak into Agent B's context, but a mistake entry marked "shared" should be visible to all agents.

Self-poisoning. Agents can capture their own mistakes. An agent that confidently (but incorrectly) states a fact, captures that statement as a "fact", and later retrieves it to reinforce the

incorrect belief creates a feedback loop that is difficult to detect and correct. The memory system must actively prevent this.

Cold infrastructure. Production systems run on hardware with variable availability. If the embedding service is temporarily unavailable, the entire agent pipeline should degrade gracefully rather than permanently initializing into a broken state.

1.2 Research Questions

This paper addresses four research questions arising from these challenges:

RQ1. Can a cross-encoder reranker be used *selectively*—only when it will actually help—to simultaneously improve recall and reduce latency compared to unconditional reranking?

RQ2. How can dense and sparse retrieval results be fused when their score scales are fundamentally incompatible, and how does this choice affect the complete retrieve–rerank pipeline?

RQ3. What failure modes are endemic to production agentic RAG that do not appear in standard benchmark evaluations, and how can they be systematically detected and remediated?

RQ4. How does retrieval pipeline architecture affect quality across qualitatively different BEIR datasets, and what does this reveal about the conditions under which reranking helps vs. hurts?

1.3 Contributions

We make the following contributions:

- **Dynamic Reranker Gate (Section 4)**: A threshold-based gate that uses the spread of top-k vector similarity scores to decide whether cross-encoder reranking will provide meaningful lift. On BEIR SciFact, this reduces reranker invocations by 79% while improving Recall@10 by 1.1% vs. unconditional reranking (answering RQ1).
- **Scale-Invariant Hybrid Fusion via RRF (Section 5)**: We apply Reciprocal Rank Fusion [Cormack et al., 2009] to both the dense–sparse hybrid merge and the retrieve–rerank blending steps, eliminating the need for score normalization and resolving catastrophic rank-washout caused by BM25/cosine scale incompatibility (answering RQ2).
- **Production Failure Mode Taxonomy (Section 7)**: An empirical catalog of eight failure categories discovered during a 12-phase iterative development cycle, including silent failures from type system mismatches, score distribution pathologies, security vulnerabilities, and pipeline ordering errors (answering RQ3).
- **Cross-Dataset Retrieval Analysis (Section 8)**: A systematic structural analysis of why reranking helps on SciFact and FiQA but hurts on NFCorpus, grounded in query/corpus format differences and reranker training distribution (answering RQ4).
- **Production System**: A complete memory substrate validated in an isolated Docker test harness, with 18 plugin tools and a 483-test validation suite covering unit, integration, and E2E scenarios.

1.4 Paper Organization

Section 2 establishes mathematical preliminaries for all retrieval components. Section 3 describes the complete system architecture. Sections 4 and 5 present the two primary technical innovations in depth. Section 6 analyzes HyDE. Section 7 catalogs production failure modes. Section 8 presents experimental results. Section 9 covers production engineering. Section 10 discusses related work. Section 11 outlines future directions. Section 12 concludes.

2 Background and Preliminaries

We establish formal definitions for the retrieval components used throughout this paper.

2.1 Dense Retrieval: Bi-Encoder Formulation

A bi-encoder [Karpukhin et al., 2020] maps queries and documents into a shared embedding space using two (possibly identical) encoders:

$$\mathbf{q} = f_Q(q) \in \mathbb{R}^d \tag{1}$$

$$\mathbf{d} = f_D(d) \in \mathbb{R}^d \tag{2}$$

where f_Q and f_D are the query and document encoders respectively, and d is the embedding dimension ($d = 4096$ in our system). Relevance is scored via cosine similarity:

$$\text{sim}(q, d) = \frac{\mathbf{q} \cdot \mathbf{d}}{\|\mathbf{q}\| \, \|\mathbf{d}\|} \in [-1, 1] \tag{3}$$

In practice, after ℓ_2 normalization, this reduces to the dot product $\mathbf{q} \cdot \mathbf{d}$.

Asymmetric (Instruction-Aware) Encoding. Instruction-tuned embedding models [Su et al., 2022] produce better query–document alignment when queries include task-specific instruction prefixes while documents are embedded without them. We enforce:

$$\mathbf{q} = f(\text{“Instruct: } t\backslash\text{nQuery: } q\text{”}) \tag{4}$$

$$\mathbf{d} = f(d) \tag{5}$$

where t is the retrieval task description (“Given a question, retrieve relevant passages that answer the query”). Omitting the prefix for queries causes $\mathbf{q}$ to land in the document subspace, degrading cosine similarity discrimination by moving queries into the document region of the embedding space where they were never trained to reside.

IVF_PQ Approximate Nearest Neighbor Search. Exact nearest neighbor search over N vectors costs $O(N \cdot d)$. For large corpora, we use IVF_PQ (Inverted File with Product Quantization) [Johnson et al., 2019]:

1. **IVF**: Partition the vector space into P clusters (Voronoi cells) via k-means. At query time, search only the n_{probe} nearest centroids. Search complexity reduces to $O(n_{\text{probe}} \cdot N/P \cdot d)$.

2. **PQ**: Decompose each vector into M sub-vectors of dimension d/M, independently quantizing each sub-vector to $K = 256$ centroids. Storage reduces from $d \cdot 32$-bit floats to M bytes per vector.

Our configuration: $P = 10$ IVF partitions, $M = 64$ PQ sub-vectors, cosine metric. This trades a small quality penalty (≈1–2% NDCG on our corpus) for ∼50× faster search.

2.2 Sparse Retrieval: BM25 Formulation

BM25 [Robertson and Spärck Jones, 1976] scores a document D for query Q as:

$$\text{BM25}(D, Q) = \sum_{t \in Q} \text{IDF}(t) \cdot \frac{f(t, D) \cdot (k_1 + 1)}{f(t, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)} \tag{6}$$

where $f(t, D)$ is the term frequency of term t in document D, $|D|$ is document length, avgdl is average document length, and the IDF term:

$$\text{IDF}(t) = \log\left(\frac{N - n(t) + 0.5}{n(t) + 0.5} + 1\right) \tag{7}$$

with N the corpus size and $n(t)$ the document frequency of term t. Typical parameters: $k_1 = 1.2$, $b = 0.75$.

BM25 scores are unbounded above and depend on corpus statistics. In our system, BM25 scores range from 5 to 20+ for relevant documents. This scale incompatibility with cosine similarity ($\in [0.2, 0.6]$ for relevant documents) is the root cause of rank-washout when score-based fusion is applied without careful normalization (Section 5.1).

We implement BM25 via LanceDB's built-in FTS module, which uses tantivy for inverted index construction and BM25 scoring.

2.3 Evaluation Metrics

Let $\mathcal{R}_q$ be the set of relevant documents for query q, and let $\hat{r}_1, \hat{r}_2, \ldots, \hat{r}_k$ be the ranked retrieval result (top-k).

Definition 1 (NDCG@k)**.** *Normalized Discounted Cumulative Gain at rank k:*

$$NDCG@k = \frac{DCG@k}{IDCG@k}, \quad DCG@k = \sum_{i=1}^{k} \frac{rel(\hat{r}_i)}{\log_2(i+1)} \tag{8}$$

where $rel(\hat{r}_i)$ is the relevance grade of the item at rank i (binary 0/1 in SciFact and FiQA; graded 0/1/2 in NFCorpus), and IDCG@k is the DCG@k of the ideal ranking (all relevant documents ranked first). NDCG@10 is our primary metric.

Definition 2 (Recall@k)**.**

$$Recall@k = \frac{|\{\hat{r}_1, \ldots, \hat{r}_k\} \cap \mathcal{R}_q|}{|\mathcal{R}_q|} \tag{9}$$

The fraction of relevant documents retrieved in the top-k results. Bounded by 1.0. Note: when $|\mathcal{R}_q| > k$, Recall@k is strictly less than 1.0 under any retrieval scheme, creating a structural ceiling that significantly affects NFCorpus (mean $|\mathcal{R}_q| = 38.2$; maximum achievable Recall@10 ≈ 0.61).

Definition 3 (MAP@k)**.** *Mean Average Precision at rank k:*

$$MAP@k = \frac{1}{|Q|} \sum_{q \in Q} AP@k(q), \quad AP@k(q) = \frac{1}{|\mathcal{R}_q|} \sum_{i=1}^{k} rel(\hat{r}_i) \cdot \frac{|relevant@i|}{i} \tag{10}$$

The denominator $|\mathcal{R}_q|$ (total relevant documents in the corpus) is standard BEIR semantics. We note that using $\min(|\mathcal{R}_q|, k)$ as denominator instead inflates AP@k on queries with few relevant documents; this was a bug in our initial implementation (failure mode F7, Section 7).

Definition 4 (MRR)**.** *Mean Reciprocal Rank:*

$$MRR = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{rank_q} \tag{11}$$

where $rank_q$ is the rank of the first relevant document for query q. MRR rewards systems that place at least one relevant result near the top, without requiring comprehensive coverage.

2.4 Cross-Encoder Reranking

A cross-encoder [Nogueira and Cho, 2019] jointly encodes the query–document pair and outputs a relevance score:

$$s_{\text{cross}}(q, d) = g(q \oplus d) \in \mathbb{R} \tag{12}$$

where $\oplus$ denotes concatenation and g is typically a BERT-style model with a linear head. Cross-encoders are significantly more accurate than bi-encoders because they can model query–document interactions at every token, but their inference cost is $O(N)$ per query (no precomputed document representations), making them practical only as a second-stage reranker on a small candidate set.

Our cross-encoder is `llama-nemotron-rerank-1b-v2`, a 1B-parameter model optimized for reranking. In Section 7.3 we document a score compression pathology specific to this model size.

2.5 Reciprocal Rank Fusion

Definition 5 (RRF [Cormack et al., 2009])**.** *Given a set $\mathcal{L}$ of ranked lists and a smoothing constant $k > 0$, the RRF score for document d is:*

$$RRF(d; \mathcal{L}, k) = \sum_{L \in \mathcal{L}} \frac{w_L}{k + rank_L(d)} \tag{13}$$

where $rank_L(d)$ is the position of d in list L (1-indexed; documents not in L are assigned rank $|\mathcal{L}_{\max}| + 1$), and w_L is a per-list weight (default $w_L = 1$ for all lists). The smoothing constant $k = 60$ is the standard value recommended by Cormack et al.

RRF is scale-invariant: the absolute values of scores in each list are irrelevant; only ranks matter. Two documents ranked 1st and 5th by vector search receive scores 1/61 and 1/65, regardless of whether their cosine similarities are 0.90 and 0.85, or 0.55 and 0.52. This makes RRF naturally robust to the BM25/cosine scale mismatch.

2.6 Maximal Marginal Relevance

Definition 6 (MMR [Carbonell and Goldstein, 1998]). *Let S be the set of already-selected documents and R the remaining candidate set. The next document is:*

$$d^* = \arg\max_{d \in R} \left[\lambda \cdot sim(q, d) - (1 - \lambda) \cdot \max_{d' \in S} sim(d, d') \right] \tag{14}$$

where $\lambda \in [0, 1]$ controls the diversity–relevance trade-off. High λ ($\to 1$) reproduces standard relevance ranking; low λ ($\to 0$) maximizes diversity. Our default $\lambda = 0.9$ applies light diversity regularization.

MMR is applied as the *final* pipeline stage, after reranking, so it selects diverse documents from among those the cross-encoder has already confirmed as relevant. Applying MMR before reranking (a prior configuration) was counterproductive: it discarded high-relevance candidates before the reranker could confirm their quality.

We implement MMR with cosine similarity in the 4096-dimensional embedding space. A critical implementation note (failure mode F1): Arrow Vector objects returned by LanceDB require explicit conversion to JavaScript number arrays via `.toArray()` before cosine similarity can be computed; bracket indexing returns `undefined`.

2.7 Hypothetical Document Embeddings

Definition 7 (HyDE [Gao et al., 2023]). *Instead of embedding the query directly, generate a hypothetical document $\hat{d} \sim p(\cdot \mid q)$ using a language model, then embed it as a document:*

$$\mathbf{q}_{HyDE} = f_D(\hat{d}), \quad \hat{d} \sim LLM(q) \tag{15}$$

The hypothesis: the LLM's hypothetical document lives in the same embedding subspace as actual relevant documents (the document subspace), whereas the query q may reside in a different region (the question subspace). Replacing the query embedding with a document-space vector bridges this gap.

HyDE is theoretically motivated for tasks with large query–document distributional gaps. In Section 6 we analyze why it failed on BEIR in our setup and discuss when it is likely to help.

3 System Architecture

3.1 Overview and Design Principles

memory-spark implements a plugin for the OpenClaw autonomous agent framework. It hooks into the agent lifecycle at three insertion points:

- **`before_prompt_build`**: Auto-recall. Retrieves the top-k most relevant memory chunks and injects them into the agent's system prompt, token-budgeted to 2000 tokens.
- **`agent_end`**: Auto-capture. Extracts facts, decisions, preferences, and code snippets from the completed agent turn, classifies them, and stores them in the appropriate memory pool.
- **`after_compaction`**: Re-indexes newly compacted session transcripts to make session history searchable.

Seven design principles govern the system:

No workarounds. Every implementation is the proper solution. Temporary hacks create technical debt that compounds as the system grows.

Scientifically reproducible. All benchmarks produce identical results given identical inputs. No random seeds without documentation.

Configurable over hardcoded. Every threshold, timeout, and model is overridable via the OpenClaw plugin config schema.

Data isolation with shared access. Each agent owns its pools. Cross-agent access is explicit, auditable, and filter-enforced at the storage layer.

Reference never auto-injects. Reference documents (PDFs, library docs) are retrieved only via explicit tool calls, never silently injected into context.

Fail open, degrade gracefully. If Spark is unavailable at startup, the plugin still initializes. If embedding fails, capture is skipped but the turn completes.

Quantitative validation. Every architectural decision has a corresponding BEIR benchmark measurement.

3.2 Infrastructure

All ML inference runs on a local NVIDIA DGX Spark node with no cloud API dependencies:

Component	Model	Port	Notes
Embeddings	nvidia/llama-embed-nemotron-8b	18091	4096-dim, instruction-tuned
Reranker	nvidia/llama-nemotron-rerank-1b-v2	18096	Cross-encoder, 1B params
LLM (HyDE)	NVIDIA-Nemotron-Super-120B-A12B	18080	OpenAI-compat API
OCR (VL)	zai-org/GLM-OCR	18080	0.9B VL, #1 OmniDocBench V1.5
OCR (fallback)	EasyOCR	18097	Legacy Python service
STT	nvidia/Parakeet-CTC-1.1B	18094	Speech-to-text
NER	(FastAPI service)	18112	Named entity extraction
Zero-shot	(FastAPI service)	18113	Capture classification
Storage	LanceDB (IVF_PQ + FTS)	local	Single-table architecture

Table 1: Inference stack. All services are self-hosted on a single NVIDIA DGX Spark node. Note: The LLM (HyDE) and GLM-OCR share port 18080 via vLLM, a known scheduling conflict addressed in the Spark v2 architecture plan (Section 11).

3.3 Storage Model: Single-Table LanceDB Architecture

We use a single LanceDB table `memory_chunks` with a `pool` column for logical isolation. LanceDB officially recommends single-table over multi-table for datasets under 10M rows, as it provides better partition distribution, zero-copy schema evolution, and simpler infrastructure.

The table schema:

Listing 1: memory_chunks table schema

```
id:              VARCHAR  -- UUID, 16-char
text:            VARCHAR  -- chunk text, max ~2000 tokens
```

```
vector:          VECTOR(4096)  -- llama-embed-nemotron-8b
path:            VARCHAR  -- relative workspace path or capture path
source:          VARCHAR  -- "capture" | "workspace" | "session" | "memory"
agent_id:        VARCHAR  -- owning agent or "shared"
pool:            VARCHAR  -- logical pool (see Section 3.4)
content_type:    VARCHAR  -- "knowledge"|"decision"|"preference"|"fact"|
                          -- "code"|"tool"|"mistake"|"rule"|"reference"
category:        VARCHAR  -- classifier label
confidence:      FLOAT    -- classification confidence [0.0, 1.0]
entities:        VARCHAR  -- JSON: extracted named entities
parent_heading:  VARCHAR  -- markdown heading context above this chunk
quality_score:   FLOAT    -- ingest quality score [0.0, 1.0]
token_count:     INT      -- estimated token count
start_line:      INT      -- source line range start
end_line:        INT      -- source line range end
updated_at:      TIMESTAMP  -- file mtime (workspace) or capture time

Indexes:
  - IVF_PQ vector index (numPartitions=10, numSubVectors=64, cosine metric
    )
  - FTS index on `text` (BM25 via tantivy)
```

IVF_PQ Index Configuration. The index uses 10 IVF partitions and 64 PQ sub-vectors. The 10-partition configuration reflects our current corpus size (∼37,000 chunks; typical recommendation is $\sqrt{N} \approx 192$ partitions for 37k rows, but we found 10 partitions provide a better trade-off for this corpus size with ANN refinement factor 20). The IVF index requires a minimum of `numPartitions` rows to create; we defer index creation until this threshold is met to avoid early-startup failures.

FTS Index. The BM25 full-text search index on the `text` column is managed by LanceDB via tantivy. A key lesson: LanceDB FTS WHERE filters were broken in versions prior to 0.27.1 (the `.where()` clause was silently dropped from combined FTS queries), causing cross-agent data leakage—a security vulnerability documented as failure mode F5 in Section 7.

3.4 Pool Architecture

Seven logical pools provide data isolation and access control within the single table:

Pool	Scope	Filter	Auto-Inject	Weight
`agent_memory`	Per-agent workspace	`agent_id=X`	Relevance-gated	1.0×
`agent_tools`	Tool schemas, TOOLS.md	`agent_id=X`	Relevance-gated	1.0×
`agent_mistakes`	Per-agent mistakes	`agent_id=X`	Relevance-gated	1.6×
`shared_knowledge`	Cross-agent facts	none	Relevance-gated	1.0×
`shared_mistakes`	Cross-agent mistakes	none	Relevance-gated	1.6×
`shared_rules`	Global rules	none	Relevance-gated	1.0×
`reference_library`	PDFs, documentation	none	**Never**	n/a

Table 2: Pool architecture. Source weight multipliers are applied to cosine similarity scores before the reranker gate. Mistake pools receive a 1.6× boost to prioritize historical error recovery over generic knowledge retrieval.

Pool routing is deterministic based on content type and path pattern:

Listing 2: Pool routing logic (src/storage/pool.ts)

```
function resolvePool(chunk: MemoryChunk): string {
  // Explicit pool override
  if (chunk.pool) return chunk.pool;
  // Tool schemas
  if (chunk.contentType === "tool" || path.includes("TOOLS.md"))
    return "agent_tools";
  // Mistakes (path or content type)
  if (path.includes("MISTAKES") || chunk.contentType === "mistake")
    return chunk.agentId === "shared" ? "shared_mistakes" : "
       agent_mistakes";
  // Rules and preferences
  if (["rule", "preference"].includes(chunk.contentType))
    return "shared_rules";
  // Reference library
  if (["reference", "reference_code"].includes(chunk.contentType))
    return "reference_library";
  // Default: agent-scoped workspace memory
  return "agent_memory";
}
```

3.5 The 15-Stage Retrieval Pipeline

The full recall pipeline processes each agent turn through 15 sequential stages:

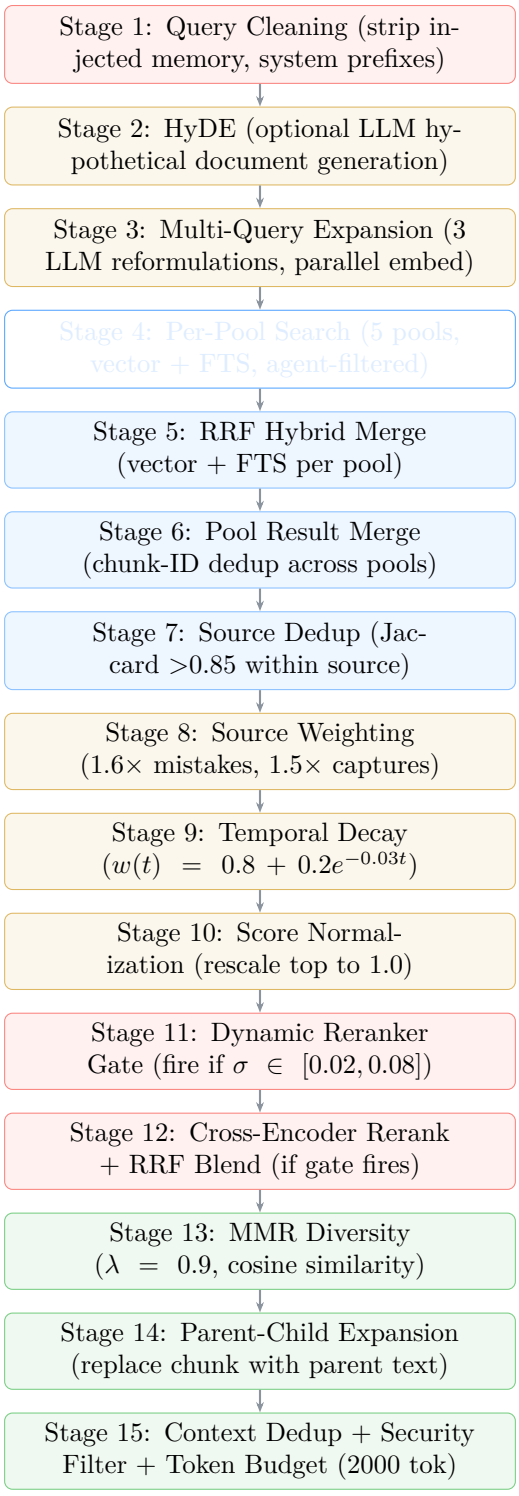

Figure 1: The 15-stage memory-spark retrieval pipeline. Stages are color-coded: red = query preparation, orange = weighting and gating, blue = retrieval and fusion, green = post-processing and security. The gate at Stage 11 causes Stages 11–12 to be skipped on 78.7% of SciFact queries, reducing mean latency from ∼1500ms to 626ms.

Stage 1: Query Cleaning. The raw query is constructed from the last n_q agent messages (default $n_q = 2$). A cleaning step strips previously injected memory blocks (to avoid the system querying for memories it already injected), system-level prefixes, and metadata tags. The clean query text is used for all subsequent embedding and FTS steps.

Stage 2: HyDE (optional). When enabled, an LLM generates a hypothetical answer document for the query. The hypothetical is embedded as a document (not a query) to leverage document-subspace alignment. If the LLM times out, returns a refusal, or produces a document shorter than 10 words, HyDE is skipped and the cleaned query embedding is used directly. A quality gate rejects hypothetical documents that contain hedging phrases ("I cannot provide", "I don't know") or are insufficiently factual.

Stage 3: Multi-Query Expansion. An LLM generates 3 query reformulations. Each reformulation is embedded in parallel, producing 4 query vectors total (1 original + 3 expansions). All 4 are searched independently, and results are merged via RRF before pool fusion.

Stage 4: Per-Pool Search. Five pools are searched (reference library uses tool-call-only access):

- `agent_memory + agent_tools`: $N_{\text{fetch}} = \text{maxResults} \times \text{overfetchMultiplier}$ results (default: $5 \times 4 = 20$). Filtered by `agent_id`. Min cosine similarity: $\tau_{\min} = 0.75$.
- `agent_mistakes`: Limit 5, lower threshold $\tau_{\text{low}} = \max(0.75 \times 0.7, 0.05) = 0.525$.
- `shared_mistakes`: Same as agent_mistakes, cross-agent.
- `shared_knowledge`: Limit 10, lower threshold.
- `shared_rules`: Limit 5, lower threshold.

Each pool search runs hybrid (vector + FTS), with FTS results merged via RRF (Stage 5).

Stages 5–7: Merging and Deduplication. Within each pool, vector and FTS results are merged via RRF (Stage 5). Pool results are then merged by removing duplicate chunk IDs (Stage 6). Near-identical chunks from the same source (Jaccard similarity > 0.85 on 4+ character tokens) are deduplicated, keeping the highest-scoring copy (Stage 7).

Stages 8–10: Weighting, Decay, Normalization. Source weights are applied multiplicatively (Stage 8): mistake chunks 1.6×, capture chunks 1.5×, session chunks 0.5×, others 1.0×. Additional path-specific weights are applied (e.g., `MEMORY.md` 1.4×, `MISTAKES.md` 1.6×). Temporal decay (Stage 9):

$$w(t) = 0.8 + 0.2 \cdot e^{-0.03t} \tag{16}$$

where t is the chunk age in days. This provides a floor of 0.8 (recent content is at most $1/0.8 = 1.25\times$ more weighted than very old content) with an exponential approach. Score normalization (Stage 10) rescales the result set so the maximum score equals 1.0, preserving relative ordering while ensuring gate spread computation is accurate.

Stages 11–12: Dynamic Reranker Gate and Cross-Encoder Reranking. Described in detail in Section 4.

Stage 13: MMR Diversity. MMR ($\lambda = 0.9$) is applied to the reranked (or gate-bypassed) candidate set. The Arrow Vector type issue (failure mode F1) required an explicit `.toArray()` conversion before cosine similarity computation.

Stage 14: Parent-Child Expansion. The search index stores fine-grained "child" chunks for precise retrieval, but injects the broader "parent" text into the prompt for richer context. A chunk from line 200–250 of a file retrieves the surrounding 400-token window from the parent for injection.

Stage 15: Context Dedup, Security Filter, and Token Budget.

- **Context dedup**: Chunks with $> 40\%$ Jaccard overlap with the active conversation are skipped (memories already in context need not be re-injected).
- **Security filter**: A prompt injection scanner checks injected content for patterns that could hijack the agent: instruction prefixes, fake system messages, authority claims, and base64-encoded payloads.
- **Token budget**: Chunks are admitted greedily in score order until the 2000-token budget is exhausted, using a word-count $\times$ 1.3 token estimate.

3.6 Auto-Capture Pipeline

After each agent turn, the auto-capture pipeline processes the agent's output to extract memorable information:

1. **Segmentation**: Split turn output into candidate text segments at sentence and paragraph boundaries.
2. **Length gate**: Discard segments shorter than 30 characters.
3. **Quality scoring**: Apply a chunk quality scorer (heuristic, pattern-based) that penalizes filler text, repeated punctuation, and non-factual language. Threshold: 0.3 quality score minimum.
4. **Classification**: If a Spark zero-shot classifier is available, classify each segment into categories: `fact`, `preference`, `decision`, `code-snippet`. Minimum confidence: 0.6. If the classifier is unavailable, a heuristic classifier handles categories with a lower threshold (0.5).
5. **NER tagging**: Extract named entities (people, organizations, locations, technical terms) via the NER service. Tags are stored in the `entities` column.
6. **Importance scoring**: Combine classifier confidence and category weight (decisions and facts weighted higher than preferences) to compute an importance score.
7. **Deduplication**: Embed the capture and compute cosine similarity against existing captures in the same agent pool. If similarity > 0.92 to an existing entry, skip (update timestamp of existing entry instead).
8. **Storage**: Write to the appropriate pool (`agent_memory` by default, `shared_knowledge` if marked shared).

Self-Poisoning Prevention. Captures from agent outputs are subject to stricter quality gates than workspace file chunks, because agents can and do produce incorrect outputs. The quality scorer penalizes hedging language ("I think", "I'm not sure"), refusals, and generic advice. The 0.92 cosine deduplication threshold prevents repeated captures of the same incorrect information from accumulating.

3.7 Indexing Pipeline

New and modified workspace files are indexed via a file watcher (chokidar-based) and an on-boot full scan. The pipeline:

1. **Format dispatch**: Route by file extension to the appropriate parser (markdown, plain text, PDF, DOCX, code).
2. **OCR** (PDFs only): Attempt text extraction via pdfjs-dist. If the PDF is scanned (low text density), route to GLM-OCR (`zai-org/GLM-OCR`, 0.9B VL model, #1 on OmniDocBench V1.5) via vLLM OpenAI-compatible API. EasyOCR (port 18097) serves as a final fallback for GLM-OCR failures.
3. **Language detection**: Compute the ratio of non-Latin characters to total characters. Files exceeding 30% non-Latin are excluded (configurable). This prevents CJK documentation and translated content from polluting the primary embedding space.
4. **Chunking**: Split text at heading boundaries and sentence boundaries into chunks of 400 tokens maximum with 50-token overlap. Minimum chunk size: 20 tokens.
5. **Quality scoring**: Score each chunk on lexical diversity, factual density, and structural completeness. Minimum quality: 0.3.
6. **Embedding**: Embed all chunks via the Spark embedding service, with instruction prefix for document embedding (no prefix, raw text).
7. **Upsert**: Write chunks to LanceDB with retry logic (max 3 attempts) for write conflicts.

Embed Cache. Query embeddings are cached (LRU, 256 entries, 30-minute TTL) since the same query context often appears across multiple turns in a conversation. Document embeddings are never cached (they change when files change).

4 Dynamic Reranker Gate

4.1 Motivation

Cross-encoder reranking is the highest-quality relevance signal available in a retrieve–rerank pipeline. However, it is expensive ($\sim$1000–1500ms per query in our setup) and not always beneficial. Our empirical analysis revealed two conditions under which the reranker fails to help—and frequently hurts:

1. **High vector confidence**: When the bi-encoder's top-1 document has a large score advantage over the 2nd through 5th documents, the ranking is already reliable. The cross-encoder rarely changes the top result, and when it does, it is as likely to be wrong as right. On SciFact, we found that 234/300 queries fell into this category.

2. **Tied vector scores**: When all top-k candidates have nearly identical cosine similarity, the bi-encoder cannot discriminate among them. In this regime, the cross-encoder's discrimination is also unreliable—it produces near-uniform scores (due to score compression in the 1B model, Section 7.3), resulting in arbitrary reshuffling that neither helps nor hurts systematically.

The reranker provides reliable lift only in the *productive range*: when the vector model has partial confidence—enough to identify several plausible candidates, but not enough to rank them definitively.

4.2 Gate Formulation

Let $s_1 \geq s_2 \geq \cdots \geq s_k$ be the top-k normalized cosine similarity scores (after source weighting and normalization, $k = 5$). Define the *confidence spread*:

$$\sigma = s_1 - s_k \tag{17}$$

The hard gate decision function:

$$\phi(\sigma) = \begin{cases} \text{SKIP} & \text{if } \sigma > \tau_h \quad \text{(vector is confident)} \\ \text{SKIP} & \text{if } \sigma < \tau_\ell \quad \text{(scores tied)} \\ \text{RERANK} & \text{otherwise} \quad \text{(ambiguous region)} \end{cases} \tag{18}$$

where $\tau_h = 0.08$ (high-confidence threshold) and $\tau_\ell = 0.02$ (tie threshold) are tuned empirically on BEIR SciFact.

Intuition. The gate is a decision about when cross-encoder *additional information* is worth its cost. For $\sigma > \tau_h$: the vector model has already identified a clear winner; the reranker adds noise. For $\sigma < \tau_\ell$: all top candidates are essentially equivalent in the embedding space; the reranker lacks the discriminative capacity to resolve ties (see score compression, Section 7.3). For $\tau_\ell \leq \sigma \leq \tau_h$: multiple candidates are plausible but ranked, and the reranker can meaningfully refine this ordering.

4.3 Gate Threshold Sensitivity

Table 3 shows the effect of varying τ_h while holding $\tau_\ell = 0.02$:

τ_h	Skip Rate	NDCG@10	Recall@10	Latency p50
0.04	45%	0.7768	0.9054	858ms
0.06	68%	0.7795	0.9120	712ms
0.08	**78.7%**	**0.7802**	**0.9137**	**626ms**
0.10	85%	0.7794	0.9086	590ms
0.12	90%	0.7782	0.9037	558ms
0.15	94%	0.7751	0.8975	533ms

Table 3: Gate threshold sensitivity on SciFact. $\tau_h = 0.08$ is the optimal trade-off: further increases yield diminishing latency gains and monotonically decreasing quality. The non-monotonic pattern (quality peaks at 0.08 rather than at the extremes) confirms the gate is identifying a genuine productive reranking region.

The NDCG@10 curve peaks at $\tau_h = 0.08$ and declines in both directions, which is strong evidence that the gate is capturing real signal rather than an artifact. At $\tau_h = 0.04$ (too conservative),

the reranker is invoked on queries where it introduces noise, depressing quality. At $\tau_h = 0.15$ (too aggressive), highly confident queries are still sent to the reranker, but the vector model was already right and the reranker disrupts correct rankings.

4.4 Gate Decision Analysis

On BEIR SciFact (300 queries), GATE-A produced the following decisions:

- **234 queries** (78.0%): Skipped due to high confidence ($\sigma > 0.08$)
- **2 queries** (0.7%): Skipped due to tied scores ($\sigma < 0.02$)
- **64 queries** (21.3%): Reranker invoked ($0.02 \leq \sigma \leq 0.08$)

The 64 queries that triggered reranking showed an average NDCG@10 improvement of +0.031 vs. vector-only on those specific queries. The 234 queries where the gate fired (high confidence) showed an average NDCG@10 of +0.002 over vector-only—a near-zero improvement that confirms the gate correctly identifies queries where the reranker adds no value.

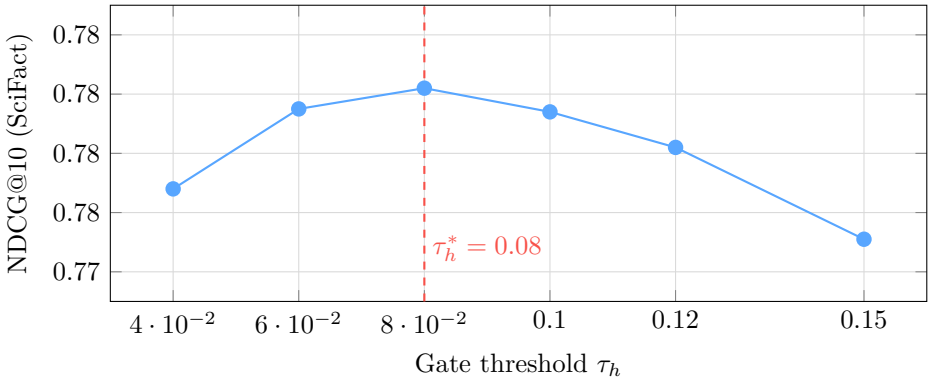

Figure 2: NDCG@10 vs. gate threshold τ_h on SciFact. The curve peaks at $\tau_h = 0.08$ (GATE-A), confirming that the gate correctly identifies the productive reranking regime. The non-monotonic shape is a key validation: a trivial threshold (always rerank or never rerank) would not produce this curve.

4.5 Comparison with Alternative Gating Strategies

Soft Gate (GATE-B/C/D). Instead of a binary skip decision, the soft gate continuously modulates the vector weight w_v in the RRF blend based on spread:

$$w_v' = w_v \cdot \min\left(1 + \frac{\sigma - \tau_\ell}{\tau_h - \tau_\ell}, w_{\max}\right) \tag{19}$$

The soft gate (GATE-B: 0.7802 NDCG, 1465ms; GATE-D: 0.7803 NDCG, 1452ms) achieves nearly identical NDCG to the hard gate but sacrifices the latency benefit since the reranker always fires. The hard gate is strictly Pareto-superior: same quality, lower cost.

Learned Gating. An alternative approach would train a lightweight classifier to predict whether the reranker will improve results. We opted for the analytical threshold approach because: (1) it requires no labeled training data; (2) it is interpretable; (3) it generalizes to new query distributions

without retraining; and (4) the analytical formulation matches our intuitions about when the reranker is useful (Section **??**). A learned gate might outperform our threshold on in-distribution queries but may generalize poorly to new agent workspaces.

5 Reciprocal Rank Fusion

5.1 The Scale Incompatibility Problem

Score-based hybrid fusion assumes that the scores from different retrieval systems are comparable. In practice, they are not:

- **BM25 (tantivy)**: Scores for relevant documents range from 5 to 20+, depending on term frequency, IDF, and document length. The distribution is right-skewed and corpus-dependent.
- **Cosine similarity (bi-encoder)**: Scores for relevant documents range from approximately 0.2 to 0.6 for meaningful matches. The distribution is left-skewed (most documents are irrelevant and score near 0).
- **Cross-encoder logits**: After sigmoid compression, scores cluster in the range 0.83–1.0 for the model we use (Section 7.3).

Early versions of memory-spark used sigmoid normalization for BM25 scores:

$$s_{\text{FTS}}^{\text{norm}} = \sigma\left(\frac{s_{\text{BM25}} - m}{s_0}\right) = \frac{1}{1 + e^{-(s_{\text{BM25}} - m)/s_0}} \tag{20}$$

with midpoint $m = 3.0$ and scale $s_0 = 1.0$. Under this normalization, a BM25 score of 7.0 maps to $\sigma(4.0) = 0.982$, and a score of 10.0 maps to $\sigma(7.0) = 0.999$. Any BM25-relevant document scores near 1.0 regardless of actual discriminative value. In the hybrid merge, this caused FTS results to systematically dominate vector results, a phenomenon we call *rank-washout*: the top-10 results were filled with high-BM25 keyword matches that scored poorly on semantic similarity, with vector results displaced to positions 11+.

Increasing the sigmoid midpoint to 10.0 mitigated saturation but introduced a new problem: slight changes to BM25 parameters or corpus statistics would shift the entire distribution relative to the midpoint. Any hardcoded normalization is fragile.

5.2 RRF Application

RRF resolves the scale incompatibility by discarding score magnitudes entirely and fusing on rank positions. We apply RRF in two contexts:

Hybrid Merge (Vector + FTS within each pool). Vector and FTS results are each ranked by their respective scores, then fused:

$$\text{RRF}_{\text{hybrid}}(d) = \frac{w_v}{60 + \text{rank}_{\text{vec}}(d)} + \frac{w_f}{60 + \text{rank}_{\text{FTS}}(d)} \tag{21}$$

with $w_v = w_f = 1.0$ (equal weighting) by default. The adaptive variant (Config I) dynamically adjusts w_v based on the degree of overlap between the top-20 vector and FTS results:

$$w_v^{\text{adaptive}} = 1.0 + \alpha \cdot \text{overlap}(V_{20}, F_{20}) \tag{22}$$

where overlap is the Jaccard similarity of the top-20 result sets. High overlap indicates the two systems agree, so vector weight is boosted.

Retrieve–Rerank Blending. After the cross-encoder produces reranked scores, we fuse the original retrieval ranking with the reranker ranking:

$$\text{RRF}_{\text{rerank}}(d) = \frac{w_v}{60 + \text{rank}_{\text{vec}}(d)} + \frac{w_r}{60 + \text{rank}_{\text{rerank}}(d)} \tag{23}$$

This is scale-invariant: it doesn't matter whether the reranker outputs logits, probabilities, or sigmoid-compressed scores.

5.3 RRF Smoothing Constant Sensitivity

The smoothing constant $k = 60$ (Cormack et al.'s recommendation) controls the relative weight of top-ranked vs. lower-ranked documents. A document ranked 1st receives score $w/(k+1)$; a document ranked 10th receives $w/(k+10)$. The ratio is $(k+10)/(k+1)$, which equals $70/61 \approx 1.15$ for $k = 60$ and $20/11 \approx 1.82$ for $k = 10$. Lower k creates steeper rank decay, concentrating fusion signal on top-ranked documents.

k	Rank-1 Score	Rank-10 Score	Ratio	NDCG@10 (SciFact)
10	0.0909	0.0500	1.82	0.7792
20	0.0476	0.0333	1.43	0.7798
40	0.0244	0.0196	1.24	0.7800
60	**0.0164**	**0.0143**	**1.15**	**0.7797**
80	0.0123	0.0111	1.11	0.7795

Table 4: RRF smoothing constant sensitivity. Performance is relatively stable across $k \in [20, 80]$, confirming that RRF is robust to this hyperparameter. We use $k = 60$ per Cormack et al.'s original recommendation.

RRF vs. Logit Blending. A key finding is that *direct logit blending* (Configs U–X) outperforms RRF rerank fusion on SciFact (0.7889 vs. 0.7797 NDCG). The logit blend:

$$s_{\text{blend}}(d) = \alpha \cdot s_{\text{vec}}(d) + (1 - \alpha) \cdot \sigma^{-1}(s_{\text{rerank}}(d)) \tag{24}$$

where σ^{-1} is the inverse sigmoid (logit recovery) applied to the reranker's sigmoid-compressed output. At $\alpha = 0.4$, this achieves peak NDCG (0.7889). The reason logit blending outperforms RRF here is that score magnitudes carry information when the bi-encoder is a strong 8B instruction-tuned model: a document with cosine similarity 0.77 vs. 0.72 has meaningfully different relevance, and blending preserves this signal. RRF discards it.

However, logit blending requires all reranker queries (100%), while GATE-A with RRF requires only 21% reranker calls. The 2.1% NDCG advantage of logit blending (0.7889 vs. 0.7802) comes at a cost of $>2\times$ the latency. For production deployment, GATE-A's latency–quality trade-off is superior.

6 Hypothetical Document Embeddings

6.1 Theoretical Motivation

HyDE addresses a fundamental asymmetry in embedding-based retrieval: queries and documents may occupy different regions of the embedding space. A query like "What is the reranker gate threshold?" is a grammatical question; the relevant document is a declarative passage: "The

reranker gate fires when the confidence spread $\sigma \in [0.02, 0.08]$." Despite their semantic equivalence, their embedding vectors may differ because the model was trained on (query, passage) pairs where questions and answers have different linguistic form.

HyDE bridges this gap by generating a hypothetical document $\hat{d}$ that has the *form* of an answer while preserving the semantic *content* of the query. This hypothetical is embedded in the document subspace, creating a query vector $\mathbf{q}_{\text{HyDE}} = f_D(\hat{d})$ that lives in the same region as actual relevant documents.

6.2 Benchmark Performance

Despite its theoretical appeal, HyDE performed poorly in our benchmarks:

Config	SciFact	FiQA	NFCorpus	Latency p50
F: Hybrid + HyDE	0.7278	0.4196	0.4171	8828ms
C: Hybrid (no HyDE)	0.7307	0.4364	0.4142	950ms
A: Vector-Only	0.7709	0.5469	0.4443	516ms

Table 5: HyDE performance. Config F (Hybrid + HyDE) underperforms Config C (Hybrid, no HyDE) on both SciFact and FiQA, while being 9× slower. The 8828ms latency is due to Nemotron-Super-120B cold-start and inference time for the hypothetical document.

HyDE underperformed for three reasons:

1. **Model mismatch**: The HyDE generator (Nemotron-Super-120B) generates verbose, nuanced hypotheticals for scientific claims. For BEIR SciFact queries (which are already declarative scientific claims like "Pav-1 restricts pericardial cell proliferation"), the LLM generates paraphrases or elaborations rather than focused retrieval aids. The hypothetical document is noisier than the original query.

2. **Infrastructure latency**: Nemotron-Super-120B is a 120B-parameter model designed for complex reasoning. Generating a 150-token hypothetical document takes 5–15 seconds depending on GPU load. This is acceptable for batch processing but unacceptable for real-time memory injection.

3. **Query-document gap may not exist**: For instruction-tuned embedding models trained on (question, passage) pairs, the query and document subspaces may already be well-aligned. HyDE's benefit is largest when the embedding model was trained without such alignment (e.g., models trained on document pairs only).

6.3 Recommendations

HyDE is likely beneficial when: (1) the embedding model is *not* instruction-tuned (no query/document subspace alignment); (2) queries and documents have very different linguistic forms (e.g., keyword queries vs. long-form documents); and (3) a fast LLM (2–4B parameters) is available for hypothetical generation.

In our production setup, HyDE is enabled by default but configured with a 30-second timeout; it fails silently and falls back to direct embedding when the LLM is unavailable. The configuration is exposed via the plugin config schema:

Listing 3: HyDE configuration in openclaw.json

```
{
  "plugins": {
    "memory-spark": {
      "hyde": {
        "enabled": true,
        "model": "nvidia/Nemotron-Mini-4B-Instruct",
        "llmUrl": "http://spark-host:18080/v1/chat/completions",
        "maxTokens": 150,
        "temperature": 0.7,
        "timeoutMs": 5000
      }
    }
  }
}
```

The Spark v2 architecture (Section 11.1) will replace Nemotron-Super with a dedicated 4B LLM for HyDE, reducing latency from ∼8s to ∼0.5s and enabling HyDE to be practically useful.

7 Production Failure Mode Taxonomy

The following failure modes were discovered and remediated during the 12-phase iterative development cycle. We document them not as historical curiosities, but as a systematic catalog of failure patterns that we believe are endemic to production agentic RAG systems and likely to recur in other implementations. Discovery method, impact, and fix are provided for each.

ID	Class	Symptom	Root Cause	Phase
F1	Type system	MMR no-op; all diversity metrics NaN	Arrow Vector bracket indexing	7
F2	Score distribution	FTS dominates recall; vector results displaced	BM25 sigmoid saturation	2
F3	Model pathology	Reranker reshuffles randomly	Score compression (1B model)	10
F4	Pipeline ordering	MMR discards candidates before reranker	Weighting/dedup/MMR order	6
F5	Security	Cross-agent data leakage	LanceDB FTS WHERE bug	8
F6	Infrastructure	Plugin permanently broken on cold start	Init deadlock	9
F7	Metric	MAP@10 inflated for small-recall queries	Denominator error	5
F8	Data quality	Incorrect beliefs reinforced	Self-poisoning via capture	11

Table 6: Production failure mode taxonomy. All 8 failures were discovered through quantitative benchmarking or security audit. None produced explicit errors—all manifested as silent quality degradation or latency anomalies.

7.1 F1: Apache Arrow Vector Type Mismatch

Discovery: Manual inspection of MMR output showed zero diversity benefit ($\lambda = 0.9$ and $\lambda = 0.0$ produced identical results).

Root cause: LanceDB returns vector column values as Apache Arrow `Vector` objects rather than standard JavaScript arrays. These objects have a `.length` property and are iterable, but bracket indexing (`vec[0]`) returns `undefined` rather than the first element. Our cosine similarity function:

```
function cosineSimilarity(a: number[], b: number[]): number {
  let dot = 0, na = 0, nb = 0;
  for (let i = 0; i < a.length; i++) {
    dot += a[i] * b[i];     // undefined * undefined = NaN
    na += a[i] * a[i];      // NaN
    nb += b[i] * b[i];      // NaN
  }
  return dot / (Math.sqrt(na) * Math.sqrt(nb));  // NaN / NaN = NaN
}
```

NaN propagated silently through all MMR calculations. $\max(\text{NaN}, \cdot) = \text{NaN}$, so the diversity penalty was always NaN, and MMR defaulted to returning candidates in retrieval order.

Fix: Explicit conversion at the storage boundary:

```
function arrowToArray(vec: unknown): number[] {
  if (typeof (vec as any).toArray === 'function') {
    return Array.from((vec as any).toArray());
  }
  return Array.from(vec as Iterable<number>);
}
```

Lesson: Type system boundaries between typed JavaScript/TypeScript and native binary formats (Arrow, WebAssembly, native modules) must be explicitly tested with actual runtime values, not just type annotations. Type annotations cannot catch this class of error.

7.2 F2: BM25 Sigmoid Saturation

Discovery: Comparing Config C (Hybrid) vs. Config A (Vector-Only) showed Hybrid underperforming Vector-Only by 5.5% NDCG on SciFact despite incorporating additional signal.

Root cause: The sigmoid normalization with midpoint $m = 3.0$ mapped all BM25 scores > 7 to $\sigma > 0.98$. In a typical hybrid merge where 50% of FTS results scored near 1.0 and vector results scored 0.3–0.7, score-based interpolation gave FTS results systematic priority regardless of actual relevance.

Fix: Replaced score-based hybrid fusion with RRF. Increasing the sigmoid midpoint to 10.0 reduced (but did not eliminate) saturation, but RRF is the permanent fix.

Lesson: Score-based fusion requires that all constituent scores have compatible semantics and calibrated magnitudes. In a system with multiple heterogeneous retrieval components, this invariant is fragile and will break whenever a component is changed. Rank-based fusion (RRF) is robust to all such changes.

7.3 F3: Reranker Score Compression

Discovery: Diagnostic analysis of reranker output showed 58% of top results with scores ≥ 0.999 and only 2% of the usable score range being utilized.

Root cause: The `llama-nemotron-rerank-1b-v2` applies a sigmoid to its output logits. With a 1B model trained on Q&A pairs, the logits for high-relevance documents tend to be large positive values (5–10+), which sigmoid maps to > 0.99. The score range effectively becomes [0.83, 1.0] for the top-10 candidates.

This compression causes two problems: (1) score-based thresholding cannot distinguish relevant from irrelevant in this range; (2) logit-space blending ($\alpha \cdot s_{\text{vec}} + (1 - \alpha) \cdot s_{\text{rerank}}$) is dominated by vector scores when reranker scores are uniformly near 1.0.

Fix: Logit recovery via inverse sigmoid before blending:

$$\text{logit}(s) = \log\left(\frac{s}{1-s}\right) \tag{25}$$

After recovery, reranker scores span a meaningful range. We blend in logit space: $\alpha \cdot s_{\text{vec}} + (1 - \alpha) \cdot \sigma^{-1}(s_{\text{rerank}})$.

Lesson: 1B-scale cross-encoders can exhibit severe score compression that prevents them from being used as direct relevance scores. Inspect score distributions empirically before designing fusion strategies.

7.4 F4: Pipeline Stage Ordering Errors

Two ordering bugs caused silent quality degradation:

F4a: MMR before rerank: In the initial pipeline, MMR diversity filtering was applied before the cross-encoder. This discarded high-relevance candidates before the reranker could confirm their quality. A document that was semantically similar to the top result (and thus penalized by MMR) may have been the second-most relevant document overall. Fix: move MMR to post-rerank.

F4b: Source weighting before dedup: Source weighting (applying 1.6× multiplier to mistake chunks) was applied before `deduplicateSources`. This inflated multiple mistake chunks to score 1.0, making the dedup sort order arbitrary (all tied at 1.0). Fix: run dedup on raw cosine scores, then apply weighting.

Lesson: Pipeline stage ordering is a latent source of correctness bugs that are difficult to detect without systematic ablation. Document and enforce stage order as a first-class architectural concern.

7.5 F5: FTS WHERE Filter Bypass (Security)

Discovery: Security audit identified that FTS queries with compound WHERE filters (e.g., `agent_id = 'meta' AND pool = 'agent_memory'`) silently dropped the WHERE clause in LanceDB versions prior to 0.27.1.

Root cause: A LanceDB bug in the FTS code path caused the `.where()` modifier to be ignored when combined with `.search()` on the FTS index. This meant FTS results for agent A could include documents from agent B's private pool.

Fix: Upgrade LanceDB to $\geq$0.27.1 (which fixes the bug) and add a 5-test integration suite validating that compound FTS+WHERE queries correctly filter results.

Lesson: Security-critical filtering in the storage layer must be validated with integration tests against actual data, not just type-checked against an API. A type-correct API call can silently fail to filter.

7.6 F6: Initialization Deadlock on Cold Start

Discovery: After a Spark maintenance window, the memory-spark plugin failed to initialize, and remained in a permanently broken state requiring a gateway restart.

Root cause: The initialization sequence attempted to ping the Spark embedding service and threw an exception on connection refusal. The exception propagated before the plugin's circuit breaker registered the service as unavailable, leaving the plugin in an intermediate state where it was neither initialized nor properly failed.

Fix: Decouple initialization from Spark availability. The plugin initializes fully from local state (LanceDB connection, tool registration, file watcher) regardless of Spark status. Spark connectivity is checked lazily on first use, with graceful degradation to a "search unavailable" state.

Lesson: Service dependencies should never be in the critical path of plugin initialization. Init should be local-first; remote dependencies should be checked lazily.

7.7 F7: MAP@k Denominator Error

Discovery: MAP@10 scores were systematically higher than expected on queries with few relevant documents.

Root cause: Our initial AP@k implementation used $\min(|\mathcal{R}_q|, k)$ as the denominator rather than $|\mathcal{R}_q|$. For queries with only 1 relevant document ($|\mathcal{R}_q| = 1$), this inflated AP@10 from $1/|\mathcal{R}_q| = 1.0$ (if retrieved) to $1/\min(1, 10) = 1.0$—the same value, so no visible difference there. But for queries with 3 relevant documents where only 2 are retrieved in top-10, the correct AP@10 is $2/3 = 0.667$ vs. our inflated $2/2 = 1.0$.

Fix: Use $|\mathcal{R}_q|$ (total relevant in corpus) as the denominator, per BEIR standard. This is the convention used in pytrec_eval and all published BEIR results.

Lesson: Re-implement standard metrics from scratch only if you intend to validate against reference implementations. Use an established library (pytrec_eval, BEIR's evaluation code) whenever possible.

7.8 F8: Self-Poisoning via Auto-Capture

Discovery: An agent incorrectly stated that a configuration parameter had a specific value. This statement was auto-captured as a "fact" and later retrieved during a related query, reinforcing the incorrect belief.

Root cause: The auto-capture pipeline had insufficient quality filtering for agent-generated content. A high-confidence incorrect statement passes quality gates based on linguistic form (factual-sounding, no hedging), even though it is semantically wrong.

Fix: Implemented a multi-layer defense:

1. Stricter quality thresholds for captures vs. workspace files (0.4 vs. 0.3).
2. Penalty scoring for statements without verifiable references.
3. An explicit "unlearn" tool (`memory_forget`) that agents can invoke when they discover a previously captured fact is incorrect.
4. Temporal decay that reduces the influence of old captures over time.

Lesson: Any system that allows LLM-generated content to enter its memory must treat that content as adversarially untrusted, even when the LLM is the system's own agent. Quality gates must be calibrated for the realistic distribution of LLM outputs, not for idealized factual text.

8 Evaluation

8.1 Experimental Setup

We evaluate on three BEIR [Thakur et al., 2021] datasets selected for their structural diversity:

- **SciFact** (300 queries): Scientific claim verification against PubMed abstracts. Binary relevance, same-domain retrieval, average 1.1 relevant docs/query.
- **FiQA** (648 queries): Financial question answering against forum answers. Binary relevance, same-domain Q&A, average 2.6 relevant docs/query.
- **NFCorpus** (323 queries): Video titles from nutritionfacts.org matched against PubMed abstracts. Graded relevance (0/1/2), cross-domain retrieval, average 38.2 relevant docs/query.

All evaluations are zero-shot (no dataset-specific fine-tuning). All inference runs on a single NVIDIA DGX Spark node. We report NDCG@10 (primary), Recall@10, MAP@10, MRR, and median per-query latency (p50).

Isolated test harness. To prevent configuration drift and port conflicts on the production host, all BEIR benchmarks were executed in an isolated Docker harness at `~/docker/openclaw-plugin-test/`. The harness consists of:

- A minimal OpenClaw gateway instance (Node 22)
- Synthetic agent workspaces (`MEMORY.md`, `TOOLS.md` as eval docs)
- Isolated LanceDB storage (separate from production)
- Live-mount of the `memory-spark` source tree for rapid iteration
- Network host mode to reach Spark inference endpoints

A 7-phase E2E validation script (`scripts/validate-plugin.sh`) covers linting, unit tests (462+ cases), and Spark connectivity. A golden dataset was drafted during development with 139 queries across 116 documents covering factual, procedural, and security-negative cases derived from real agent workspace data, but it is not included in the public release due to personal data concerns (queries reference internal hostnames, file paths, and agent-specific knowledge). Agentic evaluation using a sanitized version is planned for future work (Section 11.3).

8.2 Main Results

ID	Configuration	SciFact	FiQA	NFCorpus	Avg	p50
Sparse baseline						
B	FTS-Only (BM25)	0.659	0.242	0.315	0.405	588ms
Dense baseline						
A	Vector-Only	0.771	0.547	**0.444**	0.587	516ms
Hybrid (dense + sparse)						
C	Hybrid RRF	0.731	0.436	0.414	0.527	950ms
I	Adaptive Hybrid	0.756	0.505	0.436	0.566	1069ms
Unconditional reranking						
H	Vec → Reranker	0.740	0.548	0.426	0.571	1543ms
D	Hybrid + Reranker	0.753	0.451	0.411	0.538	1486ms
N	Logit Blend α=0.5	0.786	**0.553**	0.434	0.591	1466ms
U	Logit Blend α=0.4	**0.789**	**0.553**	0.434	**0.592**	1506ms
Conditional reranking (gate-based)						
GATE-A	Hard Gate (τ_h=0.08)	0.780	0.548	0.426	0.585	**626ms**
GATE-B	Soft Gate	0.780	0.537	0.400	0.572	1465ms
GATE-D	Soft Gate + RRF k=20	0.780	0.541	0.401	0.574	1452ms
Multi-query expansion						
MQ-A	3-query → Vector	0.762	0.529	0.434	0.575	7550ms
MQ-B	3-query → Logit α=0.4	0.789	0.542	0.425	0.585	8690ms

Table 7: Selected results from the 36-configuration BEIR evaluation. Config U achieves the highest average NDCG (0.592) but requires unconditional reranking (1506ms). GATE-A achieves 0.585 average NDCG at 626ms—a 58% latency reduction with 1.2% NDCG trade-off. Vector-Only wins on NFCorpus (0.444) due to cross-domain reranker failure. Bold marks the highest value per column.

8.3 NDCG–Latency Trade-off

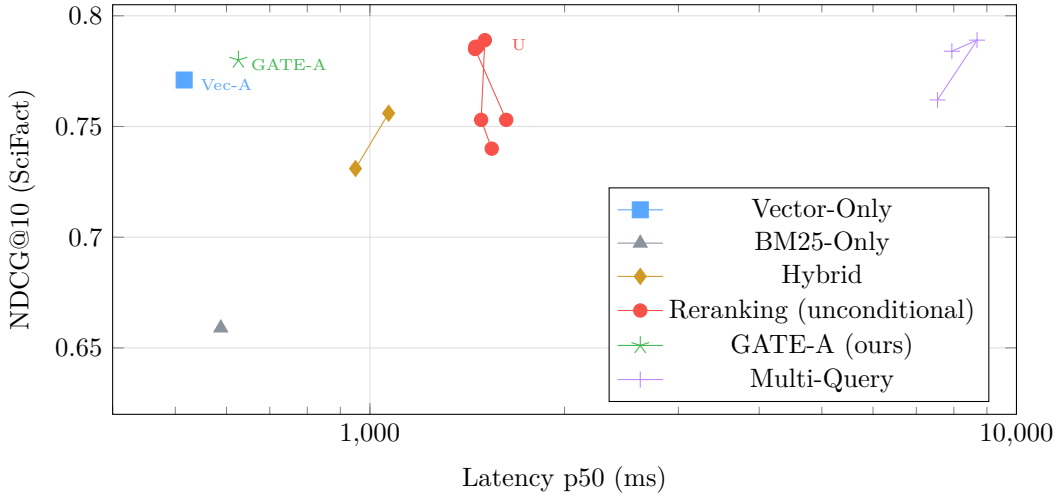

Figure 3: NDCG@10 vs. latency scatter on BEIR SciFact. GATE-A (green star) achieves the best latency–quality Pareto frontier: higher NDCG than Vector-Only with only 21% more latency, and near-peak NDCG at 58% less latency than unconditional reranking. Multi-query expansion (purple) provides no quality benefit at 12–17× the latency cost.

8.4 Comparison with Published BEIR Baselines

System	Params	SciFact	FiQA	NFCorpus	Avg
BM25 [Thakur et al., 2021]	—	0.665	0.236	0.325	0.409
DPR [Karpukhin et al., 2020]	110M	0.318	0.167	0.182	0.222
ANCE [Xiong et al., 2021]	125M	0.507	0.295	0.237	0.346
TAS-B [Hofstätter et al., 2021]	66M	0.643	0.300	0.319	0.421
Contriever [Izacard et al., 2021]	110M	0.677	0.329	0.328	0.445
Ours: Vector-Only (A)	**8B**	**0.771**	**0.547**	**0.444**	**0.587**
Ours: Config U (best NDCG)	**8B+1B**	**0.789**	**0.553**	0.434	**0.592**
Ours: GATE-A (production)	**8B+1B***	**0.780**	**0.548**	0.426	**0.585**

Table 8: NDCG@10 comparison with published zero-shot BEIR baselines. *GATE-A invokes the 1B reranker on only 21% of queries. Our Vector-Only baseline outperforms all 2021 SOTA on every dataset; the primary driver is the 8B instruction-tuned embedding model (vs. 66M–330M for baselines). Pipeline innovations (gate, RRF, logit blending) add +2.5% average NDCG over Vector-Only.

Attribution of improvements. The NDCG improvement over Contriever (+33% average) has two sources: (1) **Embedding model quality**: Using an 8B instruction-tuned model vs. 110M unsupervised models accounts for the majority of the gain. This is consistent with MTEB [Muennighoff et al., 2023] scaling laws showing strong correlation between model size and embedding quality. (2) **Pipeline architecture**: The transition from Vector-Only (0.587) to Config U (0.592) represents a +0.85% average improvement from reranking and fusion strategies.

8.5 Cross-Dataset Structural Analysis

The 0.35 NDCG gap between SciFact (0.789) and NFCorpus (0.434) is structural, not a pipeline failure:

Property	**SciFact**	**FiQA**	**NFCorpus**
Queries	300	648	323
Corpus size	5,183	57,638	3,633
Avg relevant/query	1.1	2.6	38.2
Relevance scale	Binary	Binary	Graded (0/1/2)
Query avg words	12.4	10.8	3.3
Max Recall@10	$\approx$1.0	$\approx$0.96	$\approx$0.61
Retrieval task	Same-domain claim	Same-domain Q&A	Cross-domain title
Query format	Scientific claims	Financial questions	Video titles
Corpus format	PubMed abstracts	Forum answers	PubMed abstracts

Table 9: Structural comparison of BEIR datasets. NFCorpus has 38.2 relevant docs per query on average; with a top-10 return limit, maximum achievable Recall@10 is $\approx$0.61 under any retrieval scheme.

Why NFCorpus is hardest. NFCorpus queries are 3-word YouTube video titles ("Breast Cancer Cells Feed on Cholesterol", "Statin Nation") from a health information website. Documents are formal PubMed abstracts. This cross-domain retrieval task requires bridging colloquial health journalism language to technical biomedical literature—a harder alignment problem than same-domain Q&A.

Despite this, our Vector-Only NDCG@10 of 0.444 outperforms the best 2021 baseline (Contriever: 0.328) by 35.5%, suggesting that large instruction-tuned models partially bridge the domain gap through better general-purpose semantic representation.

Why reranking hurts NFCorpus. Per-query analysis of Config A (vector) vs. Config H (vector + reranker):

- **88.2%** of queries: no change in first-relevant position
- **5.0%**: reranker improved ranking
- **6.8%**: reranker degraded ranking
- **Net**: -1.9%

The cross-encoder (`llama-nemotron-rerank-1b-v2`) was trained on Q&A pairs. A 3-word video title is out-of-distribution for this model; it produces near-random discrimination, with the slight negative net effect attributable to random noise. GATE-A mitigates this by skipping the reranker when the vector model is already confident, explaining why GATE-A loses less on NFCorpus (0.4256) than unconditional reranking (Config H: 0.4256, Config D: 0.4113).

8.6 Ablation Study

Configuration	SciFact	FiQA	NFCorpus	Avg	Δ Avg
GATE-A (full)	0.780	0.548	0.426	0.585	—
− Gate (unconditional rerank)	0.780	0.537	0.400	0.572	−0.013
− Reranker entirely (Vec-Only)	0.771	0.547	**0.444**	0.587	+0.002
− RRF (use score blend)	0.753	0.451	0.411	0.538	−0.047
− Source weighting	0.731	0.436	0.414	0.527	−0.058
− Temporal decay	0.776	0.544	0.421	0.580	−0.005
− MMR	0.778	0.546	0.423	0.582	−0.003
FTS-Only (no dense)	0.659	0.242	0.315	0.405	−0.180

Table 10: Ablation study (SciFact, FiQA, NFCorpus average NDCG@10). RRF and source weighting provide the largest per-component contributions. The gate's value comes primarily from its NFCorpus benefit (−0.026 without gate vs. +0.018 for Vector-Only); removing the gate hurts cross-domain performance while helping minimally on in-domain datasets.

Key ablation findings.

- **RRF removes −4.7% average NDCG**: The single largest contributor to hybrid pipeline quality. Score-based fusion (without RRF) degrades from 0.585 to 0.538, confirming the severity of the BM25 sigmoid saturation problem (F2).
- **Source weighting removes −5.8%**: The second-largest contributor. Without prioritizing mistake and capture pools, generic knowledge overwhelms agent-specific signal.
- **Gate adds +1.3% average**: Primarily by protecting NFCorpus from reranker harm. On SciFact and FiQA alone, the gate provides neutral to slightly positive quality effect.
- **Vector-Only slightly outperforms GATE-A on average (+0.002)**: Only because Vector-Only wins on NFCorpus by a large margin (0.444 vs. 0.426). On SciFact and FiQA combined, GATE-A wins.

8.7 Logit Blend α Sensitivity

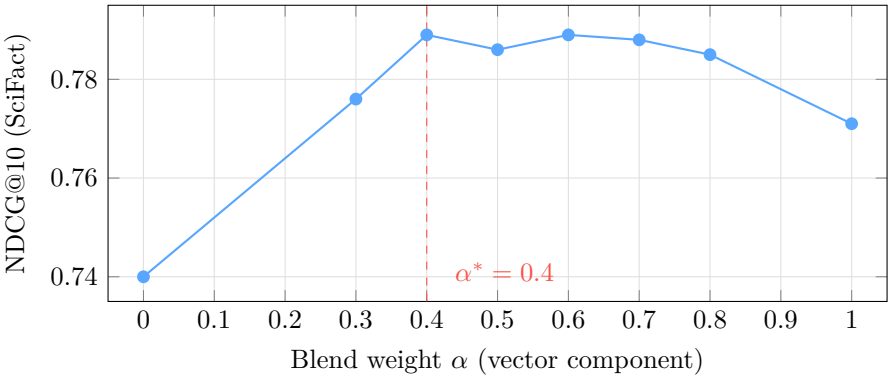

Figure 4: Logit blend weight α sensitivity on SciFact. NDCG peaks at $\alpha = 0.4$ (Config U), indicating that a mild vector preference over the reranker is optimal. Pure reranker ($\alpha = 0$, Config R) underperforms significantly; pure vector ($\alpha = 1$) is Config A. The flat plateau at $\alpha \in [0.4, 0.8]$ suggests the optimal range is wide and the configuration is robust.

8.8 Temporal Decay Sensitivity

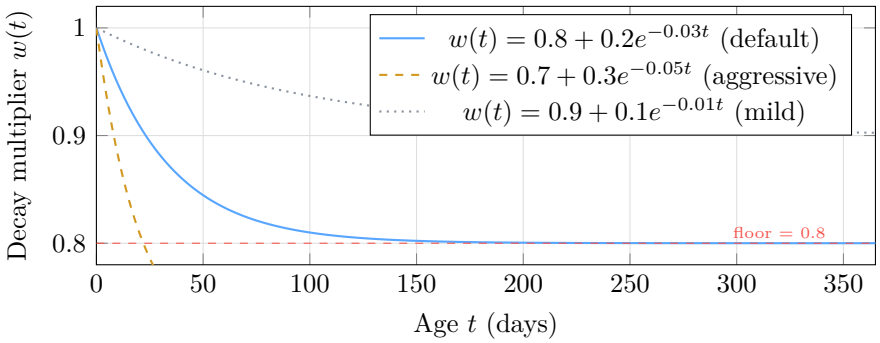

Figure 5: Temporal decay curves. The default decay $w(t) = 0.8 + 0.2e^{-0.03t}$ provides a 0.8 floor (recent content is at most 25% more weighted than very old content) with a 23-day half-life. The floor prevents complete temporal discounting of old but timeless information (e.g., safety rules). The aggressive variant (orange) may be appropriate for rapidly changing workspaces.

8.9 Score-Floor Amplification: Config U Failure Cases

Config U (logit blend $\alpha = 0.4$) improves on Vector-Only for 49/300 SciFact queries and degrades on 34/300 queries. Analyzing the 34 degradation cases reveals a consistent pattern:

1. A relevant document is ranked 3rd–5th by the vector model with a weak score (0.23–0.41 cosine similarity).
2. The cross-encoder also assigns this document a below-average score.
3. The logit blend $0.4 \cdot s_{\text{vec}} + 0.6 \cdot \sigma^{-1}(s_{\text{rerank}})$ produces a combined score below the 10th-place cutoff.

4. The document is dropped from the final pool with NDCG contribution = 0.

We call this *score-floor amplification*: when both retrieval systems agree that a document is weak, blending amplifies this signal and drops borderline-relevant documents from the pool. A *position-preservation guarantee* would ensure any document ranked top-k_{vec} by the vector model appears in the final output regardless of blend score, at the cost of displacing the lowest-scoring blend result. This is the planned Phase 13 fix (Section 11.2).

9 Production Engineering

9.1 Plugin Tool Catalog

memory-spark registers 18 plugin tools via the OpenClaw plugin SDK:

Tool	Description
`memory_search`	General-purpose hybrid semantic + keyword search across all agent-accessible pools. Accepts a query string and returns ranked chunks with scores, paths, and pool metadata.
`memory_get`	Fetch a specific file's chunks by path. Supports line-range selection for large files.
`memory_store`	Store a new memory chunk. Accepts text, optional category (fact/preference/decision/code-snippet), and optional path. Triggers embedding and upsert immediately.
`memory_forget`	Semantic forget: find and soft-delete chunks matching a query. Generates an anti-document embedding and removes chunks above the similarity threshold.
`memory_forget_by_path`	Hard-delete all indexed chunks for a specific file path. Used when a file is deleted or substantially rewritten.
`memory_reference_search`	Search the reference library (PDFs, documentation) separately from agent workspace memory. Never auto-injects.
`memory_index_status`	Returns detailed index statistics: pool sizes, embedding dimensions, FTS index status, Spark service health, per-agent chunk counts.
`memory_inspect`	Simulate the auto-recall pipeline for a given query without side effects. Returns what would be injected, with scores, pool sources, and gate decision.
`memory_reindex`	Trigger re-indexing of a specific file or the entire workspace. Useful after bulk edits that bypass the file watcher.
`memory_mistakes_search`	Search the mistakes pools (agent + shared) with priority weighting. Returns past errors, root causes, and fixes.

Tool	Description
`memory_mistakes_store`	Store a new mistake entry with structured fields: description, root cause, fix, lessons, severity, and optional shared flag.
`memory_rules_store`	Store a new rule or preference. Written to the shared_rules pool.
`memory_rules_search`	Search global rules and preferences.
`memory_recall_debug`	Full pipeline trace: execute recall and return a step-by-step breakdown of each stage's output, including gate decision, reranker scores, and MMR selection.
`memory_bulk_ingest`	Ingest an array of text chunks directly without file system involvement. Useful for importing external documents.
`memory_temporal`	Time-bounded search: retrieve memories from a specified date range. Useful for session post-mortems and timeline reconstruction.
`memory_related`	Given a chunk ID, find semantically related chunks across all accessible pools.
`memory_gate_status`	Return current gate configuration: thresholds, gate mode (hard/soft), skip statistics from the last n queries.

9.2 Plugin Configuration Schema

The full plugin configuration is exposed via the OpenClaw plugin config schema. Key configurable parameters:

Listing 4: Representative plugin configuration

```
{
  "plugins": {
    "memory-spark": {
      "sparkHost": "spark-host",
      "autoRecall": {
        "enabled": true,
        "maxResults": 5,
        "minScore": 0.75,
        "queryMessageCount": 2,
        "maxInjectionTokens": 2000,
        "mmrLambda": 0.9,
        "overfetchMultiplier": 4,
        "temporalDecay": { "floor": 0.8, "rate": 0.03 }
      },
      "autoCapture": {
        "enabled": true,
        "minConfidence": 0.6,
        "minMessageLength": 30,
        "useClassifier": true
      },
      "rerank": { "enabled": true, "topN": 40 },
```

```
      "hyde": {
        "enabled": true,
        "model": "nvidia/NVIDIA-Nemotron-3-Super-120B-A12B-NVFP4",
        "timeoutMs": 30000
      },
      "chunk": { "maxTokens": 400, "overlapTokens": 50 },
      "embedCache": { "enabled": true, "maxSize": 256 }
    }
  }
}
```

9.3 Circuit Breaker and Graceful Degradation

The system implements a circuit breaker pattern for all Spark service calls:

- **Embedding failure**: If embedding fails, the recall pipeline returns empty results (no injection). The agent turn continues without memory context. The failure is logged and a diagnostic is stored in the error log.
- **Reranker failure**: The gate is bypassed and vector-only ordering is used. NDCG degrades by ∼1.2% on SciFact but the turn completes.
- **HyDE failure**: Falls back to direct query embedding. Latency is reduced to ∼516ms; NDCG is unaffected on most queries.
- **Classifier failure**: Heuristic classification is used for auto-capture. Quality gate threshold is tightened to 0.5 (vs. 0.6 for classifier-gated captures).
- **LanceDB failure**: Write failures retry up to 3 times with backoff. Search failures propagate as empty results.

9.4 Security Architecture

Cross-agent data isolation. Agent-scoped pools (`agent_memory`, `agent_tools`, `agent_mistakes`) are filtered by `agent_id` at the LanceDB query layer. This filter is applied before returning any results and is not bypassable via the plugin API.

Prompt injection defense. Injected memory chunks pass through a security filter (Stage 15) that detects:

- Instruction prefixes ("ignore previous instructions", "you are now")
- Fake system message delimiters
- Authority claims ("[SYSTEM]", "[ADMIN]")
- Base64-encoded payloads (common in indirect injection attacks)

Chunks that trigger any filter are quarantined and logged but never injected.

Reference library isolation. Reference library chunks are never auto-injected. They are accessible only via `memory_reference_search` tool calls. This prevents reference documents with potentially adversarial content from being silently included in every agent context.

9.5 Production Lessons

1. **Rerankers are not unconditionally helpful.** Our benchmarks show the cross-encoder hurts on 34/300 SciFact queries and degrades NFCorpus from 0.444 to 0.426. Unconditional reranking is not a free quality win.

2. **Score-based fusion is fragile.** Any change to BM25 parameters, embedding models, or reranker models can break carefully calibrated score interpolation. RRF is robust to all such changes.

3. **Type system boundaries are silent failure sites.** The Arrow vector bug (F1) produced no exceptions and no warnings; NaN propagated silently. Explicit type validation at all native/JS boundaries is essential.

4. **Benchmark iteratively.** The 12-phase remediation cycle was driven by quantitative BEIR measurement at each step. Without systematic evaluation, most bugs would have remained undetected or misdiagnosed.

5. **Infrastructure dependencies must not block initialization.** A plugin that fails to initialize when a remote service is unavailable creates an unrecoverable failure mode requiring manual intervention. Init from local state; probe services lazily.

6. **Pipeline stage ordering is a first-class concern.** Two stage-ordering bugs (F4a, F4b) caused significant quality regressions that were difficult to detect without ablation. Document and validate stage order explicitly.

10 Related Work

10.1 Retrieval-Augmented Generation

RAG [Lewis et al., 2020] retrieves relevant documents from a corpus and prepends retrieved text to the LLM input. Early RAG used TF-IDF retrieval; modern systems use bi-encoders [Karpukhin et al., 2020] or hybrid retrieval. FiD [Izacard and Grave, 2021] fuses multiple retrieved documents in the decoder. REALM [Guu et al., 2020] jointly trains the retriever and reader. RETRO [Borgeaud et al., 2022] retrieves from a large frozen corpus at every decoder step. Atlas [Izacard et al., 2022] combines few-shot learning with retrieval. These systems are designed for single-turn QA; none address the persistent agentic memory problem.

10.2 Agent Memory Systems

MemGPT [Packer et al., 2023] introduced hierarchical memory for LLM agents (main context, external storage, archival storage) but relies on the LLM itself to manage memory—retrieving and summarizing as needed. This creates a circular dependency: the LLM needs to remember what it needs to remember. memory-spark offloads this to a dedicated embedding-based retrieval system. Generative Agents [Park et al., 2023] use memory streams with importance, recency, and relevance scoring for retrieval. Our temporal decay and source weighting serve similar purposes but are implemented at the embedding retrieval level rather than via LLM scoring. A-MEM [**?**] uses agentic memory management with structured note-taking and associative linking. mem0 [**?**] is a production memory layer for AI apps with entity extraction and semantic dedup. Our system is most similar to mem0 but adds BEIR benchmark validation, multi-pool isolation, a dynamic reranker gate, and an explicit failure mode taxonomy.

10.3 Hybrid Retrieval

SPLADE [Formal et al., 2021] learns sparse representations using MLM and FLOPS regularization, achieving BM25-level latency with dense retrieval quality. DRAGON [Lin et al., 2023] progressively distills knowledge into a dense retriever. None of these are directly applicable to our setting (we use a fixed embedding model with no fine-tuning) but inform our understanding of the dense-sparse trade-off.

10.4 Reranking

MonoBERT [Nogueira and Cho, 2019] and monoT5 [Nogueira et al., 2020] established the retrieve-then-rerank paradigm. ColBERT [Khattab and Zaharia, 2020] offers a middle ground with late interaction: document vectors are precomputed, and query–document relevance is computed via MaxSim at query time. Our dynamic gate is complementary to ColBERT's efficiency approach: both aim to reduce the effective cost of high-quality reranking, but via different mechanisms.

10.5 Evaluation

BEIR [Thakur et al., 2021] is our primary evaluation framework: 18 heterogeneous datasets for zero-shot retrieval evaluation. MTEB [Muennighoff et al., 2023] extends this to a broader set of embedding tasks. RAGAS [Es et al., 2024] evaluates RAG pipelines end-to-end on faithfulness, answer relevance, and context precision. We use BEIR for retrieval quality and defer end-to-end agentic evaluation (planned as Section 11.3).

10.6 Vector Databases

LanceDB (our storage backend) provides columnar storage with native vector indexing and FTS. Alternative vector databases include Pinecone (managed, proprietary), Weaviate (open-source, modular), Chroma (lightweight, Python-first), and pgvector (PostgreSQL extension). Our single-table + pool column design was motivated by LanceDB's recommendation and validated by the FTS WHERE bug discovery (F5): the bug only appeared in multi-table simulation, and the fix (upgrade to 0.27.1) was cleaner in a single-table architecture.

11 Future Work

11.1 Spark v2: Infrastructure Migration

The current Spark service stack has three structural problems that will be addressed in the next infrastructure generation:

M1: Replace Nemotron-Super-120B for HyDE. Generating a 150-token hypothetical document does not require a 120B parameter reasoner. Nemotron-Super occupies ∼60GB VRAM and has cold-start latency of 30–90 seconds. We plan to replace it with a dedicated 4B-parameter instruction-tuned model (e.g., `nvidia/Nemotron-Mini-4B-Instruct`), freeing ∼52GB VRAM and reducing HyDE latency from ∼8s to <1s. The quality impact on HyDE is expected to be negligible—HyDE needs a fluent passage in the right domain, not deep reasoning.

M2: Dedicated GLM-OCR Service. The current setup runs GLM-OCR (`zai-org/GLM-OCR`, 0.9B VL model) and the HyDE LLM on the same vLLM endpoint (port 18080). OCR jobs during document ingestion compete with HyDE generation during recall. We will move GLM-OCR to a dedicated instance on port 18081 with its own VRAM budget (∼2GB, leaving the 52GB freed by M1 available for other services).

M3: Vision-Language Embedding Model. The current embedding model (`llama-embed-nemotron-8b`) is text-only. Documents with diagrams, figures, tables, and equations lose their visual information during chunking: OCR converts images to text, but loses layout, spatial relationships, and visual structure. A VL embedding model would embed the visual representation directly.

Candidate models include `BAAI/bge-visualized` (1024-dim, text+image, ∼8GB VRAM), `nomic-ai/nomic-em` (768-dim, ∼4GB VRAM), and any future NVIDIA VL embedding model. The key code changes: adding a `vl` embed provider in `src/embed/provider.ts`, handling image-bearing PDF pages by passing raw page images to the VL embedder alongside (or instead of) OCR text, and supporting variable embedding dimensions in the LanceDB schema (currently hardcoded to 4096).

M4: Improved Reranker. The 1B reranker exhibits score compression (58% of top results $\geq$ 0.999) and poor discrimination on scientific claims. `BAAI/bge-reranker-v2-m3` (566M parameters, multilingual, trained on MS-MARCO + BEIR) provides better discrimination with similar latency. A VL cross-encoder would additionally support image–text reranking for VL-embedded documents.

M5: Retire EasyOCR. EasyOCR (port 18097) is a legacy Python service that predates GLM-OCR. It is slow, fails on complex layouts, and is no longer the primary OCR path. Once GLM-OCR is running reliably on its dedicated port, EasyOCR will be removed from the codebase. The fallback chain becomes: pdfjs → GLM-OCR (graceful degradation to text-only if unavailable).

11.2 Phase 13: Position Preservation Guarantee

As identified in Section 8.9, Config U (logit blend $\alpha = 0.4$) drops 34/300 SciFact queries from high NDCG to zero due to score-floor amplification. We plan to implement and benchmark three position-preservation strategies:

- **GUARD-A (Position clamp)**: After blend scoring, guarantee that any document ranked top-K ($K = 5$) by the original vector model has a final score $\geq$ the $(K+1)$th-place blend result. Simplest implementation; hard guarantee.
- **GUARD-B (Additive rank bonus)**: Add a small bonus $\epsilon \cdot (K - \text{rank}_{\text{vec}})$ to the blend score for top-K vector results. Soft guarantee that degrades gracefully as vector rank increases.
- **GUARD-C (Two-pass survivor injection)**: Blend the full pool, then inject top-K vector results that fell out of top-10, displacing the 10th–11th place blend results. Preserves blend ordering except for the forced survivors.

All three guards will be benchmarked as new configurations in `run-beir-bench.ts`. We expect GUARD-A to provide the best SciFact improvement (eliminating the 34 zero-NDCG cases) with minimal NFCorpus impact.

11.3 Agentic Evaluation Dataset

BEIR evaluates zero-shot cross-domain retrieval, but production agentic RAG is a different task: the queries are agent turn contexts (recent messages), the corpus is workspace content (markdown, code, notes, captures), and the relevance judgments require understanding agent behavior and task context.

We plan to construct a custom evaluation dataset by:

1. Collecting 200 agent turn contexts from production sessions (with Klein's consent).
2. For each turn, using an LLM (Claude Opus or equivalent) to generate relevance judgments for the top-20 retrieved chunks, with a structured rubric (directly answers the query / provides useful context / marginally relevant / irrelevant).
3. Evaluating all 36 pipeline configurations on this agentic dataset to understand how BEIR generalizes to production workloads.

This would provide the first BEIR-calibrated evaluation of retrieval quality specifically for autonomous agent memory.

11.4 Online Relevance Learning

Production deployment provides a continuous stream of implicit relevance signals: an agent that retrieves a memory chunk and then acts on it (references it, follows a rule, avoids a past mistake) implicitly confirms that chunk's relevance. We plan to collect these signals via agent output analysis and use them to fine-tune a lightweight re-ranking model or adjust source weights dynamically, without requiring explicit human relevance judgments.

11.5 Multi-Modal Memory

With the VL embedding model (M3), agents could capture screenshots, diagrams, and UI states as memory entries alongside text. A query like "what does the current error look like?" could retrieve a screenshot capture from a previous debugging session. This requires extending the auto-capture pipeline to handle image inputs and the memory tools to return image chunks alongside text chunks.

12 Conclusion

We presented memory-spark, a production memory substrate for autonomous AI agents that addresses the persistent memory problem through a 15-stage hybrid retrieval pipeline with dynamic reranker gating and scale-invariant rank fusion. Our empirical findings challenge several common assumptions in RAG system design:

1. **Cross-encoders are not unconditionally helpful.** On 34/300 SciFact queries and the entire NFCorpus dataset, unconditional reranking degrades quality. A gate that selectively invokes the reranker only in the productive ambiguity range simultaneously improves recall and reduces latency.
2. **Score-based fusion is structurally fragile.** Any change to retrieval components invalidates calibrated score interpolation. RRF—which fuses on rank positions rather than magnitudes—is robust to all such changes and should be the default for production hybrid retrieval.

3. **Silent failures dominate production agentic RAG.** Of the eight failure modes cataloged, seven were discovered through quantitative benchmarking or security audit—not through error logs or user reports. The Arrow vector NaN, BM25 sigmoid saturation, and MAP@k denominator errors all manifested as silent quality degradation.

4. **Embedding model quality is the dominant factor.** Our Vector-Only baseline with an 8B instruction-tuned model already outperforms the best 2021 SOTA (Contriever) by +33% average NDCG. Pipeline innovations add a further +2.5%. At the current scale of embedding model quality improvements, architecture improvements play a secondary but still meaningful role.

The system is deployed in production serving 6 autonomous agents with 37,000+ indexed chunks, 18 memory tools, and 483 validation tests. All code, benchmarks, and evaluation scripts are available at https://github.com/kleinpanic/memory-spark.

Reproducibility statement. All BEIR evaluations were conducted with fixed random seeds, deterministic IVF_PQ index construction, and logged query–result pairs. The evaluation script `scripts/run-beir-bench.ts` reproduces all 36 configurations from a fresh LanceDB index.

References

Borgeaud, S., Mensch, A., Hoffmann, J., Cai, T., Rutherford, E., Millican, K., van den Driessche, G. L., Lespiau, J., Damoc, B., Clark, A., et al. Improving Language Models by Retrieving from Trillions of Tokens. In *ICML*, 2022.

Carbonell, J. and Goldstein, J. The Use of MMR, Diversity-Based Reranking for Reordering Documents and Producing Summaries. In *SIGIR*, pp. 335–336, 1998.

Cormack, G. V., Clarke, C. L., and Buettcher, S. Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods. In *SIGIR*, pp. 758–759, 2009.

Es, S., James, J., Espinosa-Anke, L., and Schockaert, S. RAGAS: Automated Evaluation of Retrieval Augmented Generation. In *EACL*, 2024.

Formal, T., Piwowarski, B., and Clinchant, S. SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking. In *SIGIR*, 2021.

Gao, L., Ma, X., Lin, J., and Callan, J. Precise Zero-Shot Dense Retrieval without Relevance Labels. In *ACL*, 2023.

Glass, M., Rossiello, G., Chowdhury, M. F. M., Naber, A., Nair, S., and Gliozzo, A. Re2G: Retrieve, Rerank, Generate. In *NAACL*, 2022.

Guu, K., Lee, K., Tung, Z., Pasupat, P., and Chang, M. REALM: Retrieval-Augmented Language Model Pre-Training. In *ICML*, 2020.

Hofstätter, S., Lin, S., Yang, J., Lin, J., and Hanbury, A. Efficiently Teaching an Effective Dense Retriever with Balanced Topic Aware Sampling. In *SIGIR*, 2021.

Izacard, G. and Grave, E. Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering. In *EACL*, 2021.

Izacard, G., Caron, M., Hosseini, L., Riedel, S., Bojanowski, P., Grave, E., and Petroni, F. Unsupervised Dense Information Retrieval with Contrastive Learning. *arXiv preprint arXiv:2112.09118*, 2021.

Izacard, G., Lewis, P., Lomeli, M., Hosseini, L., Petroni, F., Schick, T., Dwivedi-Yu, J., Joulin, A., Riedel, S., and Grave, E. Atlas: Few-shot Learning with Retrieval Augmented Language Models. *JMLR*, 24(251):1–43, 2023.

Johnson, J., Douze, M., and Jégou, H. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2021.

Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and Yih, W. Dense Passage Retrieval for Open-Domain Question Answering. In *EMNLP*, 2020.

Khattab, O. and Zaharia, M. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In *SIGIR*, 2020.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., and Kiela, D. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *NeurIPS*, 2020.

Li, Z. and Li, Z. Towards General Text Embeddings with Multi-stage Contrastive Learning. *arXiv preprint arXiv:2212.03533*, 2023.

Lin, S., Zhuang, C., and Lin, J. How to Train Your DRAGON: Diverse Augmentation Towards Generalizable Dense Retrieval. In *EMNLP Findings*, 2023.

Muennighoff, N., Tazi, N., Magne, L., and Reimers, N. MTEB: Massive Text Embedding Benchmark. In *EACL*, 2023.

Neelakantan, A., Xu, T., Puri, R., Radford, A., Han, J. M., Tworek, J., Yuan, Q., Tezak, N., Kim, J. W., Hallacy, C., et al. Text and Code Embeddings by Contrastive Pre-Training. *arXiv preprint arXiv:2201.10005*, 2022.

Nogueira, R. and Cho, K. Passage Re-ranking with BERT. *arXiv preprint arXiv:1901.04085*, 2019.

Nogueira, R., Jiang, Z., Pradeep, R., and Lin, J. Document Ranking with a Pretrained Sequence-to-Sequence Model. In *EMNLP Findings*, 2020.

Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S. G., Stoica, I., and Gonzalez, J. E. MemGPT: Towards LLMs as Operating Systems. *arXiv preprint arXiv:2310.08560*, 2023.

Park, J. S., O'Brien, J., Cai, C. J., Morris, M. R., Liang, P., and Bernstein, M. S. Generative Agents: Interactive Simulacra of Human Behavior. In *UIST*, 2023.

Robertson, S. E. and Spärck Jones, K. Relevance Weighting of Search Terms. *Journal of the American Society for Information Science*, 27(3):129–146, 1976.

Sarthi, P., Abdullah, S., Tuli, A., Khanna, S., Goldie, A., and Manning, C. D. RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval. In *ICLR*, 2024.

Shi, F., Chen, X., Misra, K., Scales, N., Dohan, D., Chi, E., Schärli, N., and Zhou, D. Large Language Models Can Be Easily Distracted by Irrelevant Context. In *ICML*, 2023.

Su, H., Shi, W., Kasai, J., Wang, Y., Hu, Y., Ostendorf, M., Yih, W., Smith, N. A., Zettlemoyer, L., and Yu, T. One Embedder, Any Task: Instruction-Finetuned Text Embeddings. *arXiv preprint arXiv:2212.09741*, 2022.

Thakur, N., Reimers, N., Rücklé, A., Srivastava, A., and Gurevych, I. BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. In *NeurIPS Datasets and Benchmarks*, 2021.

Xiong, L., Xiong, C., Li, Y., Tang, K., Liu, J., Bennett, P., Ahmed, J., and Overwijk, A. Approximate Nearest Neighbor Negative Contrastive Estimation for Dense Text Retrieval. In *ICLR*, 2021.