Antigravity RAG: A Hybrid Local/Cloud Personal Knowledge Graph System for Lifelong Autobiographical Memory

Oleksandr Sitnikov
Independent Research
`(private deployment)`

Marcus (AI Symbiote)
Independent Research
`(private deployment)`

February 2026

Abstract

We present **Antigravity RAG**, a production-grade retrieval-augmented generation system built as a *personal exobrain* for **lifelong autobiographical memory**. The system maintains a continuously updated memory store spanning **725,521 Telegram messages** across **812 chats**, daily logs, project documentation, and structured "memory notes" with provenance. Unlike enterprise RAG pipelines optimized for static document QA, Antigravity targets *autobiographical retrieval*: answering questions about the user's own decisions, relationships, evolving technical stack, and time-dependent facts.

Antigravity is engineered for a constrained, privacy-first deployment that can run *entirely on CPU* (8 vCPU AMD EPYC, 32 GB RAM, no GPU) using local models, while retaining a resilient cascading fallback (**Anthropic API** → **Qwen API** → **local Ollama**) for availability and latency control. The system combines hybrid retrieval (dense + BM25 with Reciprocal Rank Fusion), a personal knowledge graph with associative traversal, hierarchical summaries for broad queries, and a belief-revision layer that detects supersession and contradictions to keep memory coherent over time.

Key contributions: (1) a **7-intent semantic query router** (regex + centroid embeddings) that compiles intent-specific retrieval plans; (2) **belief-revision memory notes** with lightweight truth maintenance and timeline unwinding for "evolution" queries; (3) **RAPTOR-style** hierarchical summaries and community-level distillation for abstract questions; (4) graph-based associative recall using **Personalized PageRank (PPR)** and entity briefs; (5) **CRAG-style confidence gating** with automatic query reformulation; (6) **intent-aware token budgets** and "Gold Sandwich" context packing motivated by the *Lost in the Middle* phenomenon; (7) a crash-resilient **chat digester** that converts noisy chat logs into structured profiles; (8) a two-level cache (exact-match + semantic) with intent-aware TTL and semantic invalidation; and (9) automated memory tiering/rollups for long-term compression.

Keywords: Retrieval-Augmented Generation, Personal Knowledge Graphs, Lifelong Memory, Belief Revision, Hybrid Retrieval, CPU-only Inference, Privacy.

Contents

1 Introduction

Retrieval-Augmented Generation (RAG) combines parametric language models with non-parametric memory to improve factuality and updateability [1]. Most RAG systems assume a relatively static corpus and optimize for open-domain document QA. Personal memory retrieval differs fundamentally along five axes:

- **Temporal dynamics.** Facts and decisions evolve (e.g., "we use Pinecone" → "we migrated to LanceDB"). A personal system must represent *belief evolution* and preserve provenance.
- **Multi-source fusion.** "Who is X?" may require merging chat history, profile files, atomic notes, and graph relations.
- **Associative recall.** "How are X and Y connected?" often requires multi-hop reasoning and graph traversal beyond nearest-neighbor similarity.
- **Autobiographical framing.** Queries implicitly refer to the user ("What did *we* decide?"), requiring coreference-aware rewriting and identity-aware retrieval.
- **Resource constraints.** A personal exobrain must run 24/7 on a single VPS; CPU-first latency and crash resilience are primary design constraints.

Antigravity RAG addresses these requirements by combining hybrid retrieval, a personal knowledge graph, belief-revision memory notes, hierarchical summaries, and robust post-retrieval controls. The described deployment is private (Hetzner CCX33, Helsinki) and exposes a FastAPI daemon on port 8100 (Figure 1).

Contributions. Beyond the individual components, the main contribution is an *end-to-end architecture* that makes autobiographical retrieval reliable under CPU-only constraints, while remaining robust to memory drift (superseded facts), noisy conversational data, and prompt-injection risks.

2 Related Work

Antigravity RAG draws on several research threads; we summarize the closest ones and clarify what is different in the *personal autobiographical* setting.

2.1 RAG and Dense Retrieval

The original RAG formulation [1] popularized coupling dense retrieval with generation. Retrieval quality depends strongly on the retriever: REALM integrates retrieval into LM pretraining [2], DPR provides a widely used dense passage retriever for open-domain QA [3], and ColBERT improves retrieval efficiency via late interaction [4]. Antigravity uses a pragmatic hybrid approach (dense + BM25) merged with Reciprocal Rank Fusion (RRF) [5], optimized for heterogeneous personal corpora rather than benchmarks.

2.2 Long Context, Ordering, and Hierarchical Retrieval

Long-context models are sensitive to evidence position; "Lost in the Middle" reports a U-shaped degradation where mid-context evidence is systematically underused [9]. Antigravity operationalizes this with "Gold Sandwich" ordering in its context packer: the most relevant chunks are placed at the beginning and end of the context window. For broad queries, hierarchical indexing and summarization methods such as RAPTOR [6] provide multi-resolution retrieval. Antigravity uses a RAPTOR-inspired nightly build and community-level summaries to handle abstract questions without loading the entire corpus.

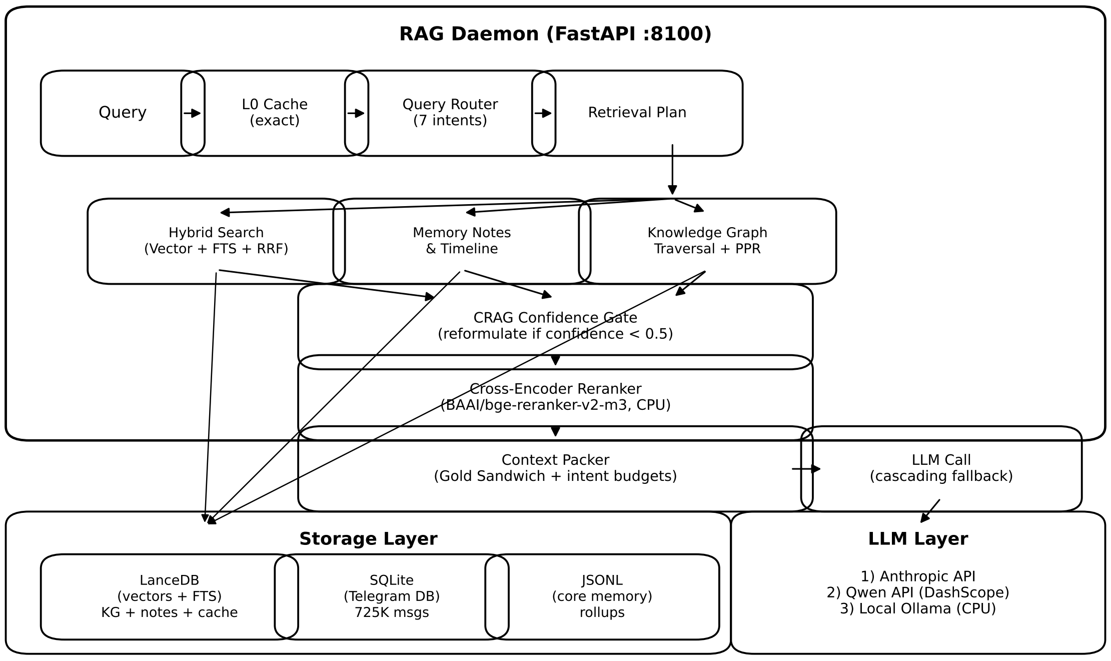
RAG Daemon (FastAPI :8100)
Query
L0 Cache (exact)
Query Router (7 intents)
Retrieval Plan
Hybrid Search (Vector + FTS + RRF)
Memory Notes & Timeline
Knowledge Graph Traversal + PPR
CRAG Confidence Gate (reformulate if confidence < 0.5)
Cross-Encoder Reranker (BAAI/bge-reranker-v2-m3, CPU)
Context Packer (Gold Sandwich + intent budgets)
LLM Call (cascading fallback)
Storage Layer
LanceDB (vectors + FTS) KG + notes + cache
SQLite (Telegram DB) 725K msgs
JSONL (core memory) rollups
LLM Layer
1) Anthropic API
2) Qwen API (DashScope)
3) Local Ollama (CPU)

Figure 1. System overview of Antigravity RAG. A query is routed into an intent-specific retrieval plan. Retrieval combines hybrid search, memory notes/timeline, and knowledge graph traversal (including Personalized PageRank). A CRAG-style confidence gate triggers reformulation on low-confidence retrieval. Results are reranked with a CPU-friendly cross-encoder and packed with intent-aware budgets and Gold Sandwich ordering before the final LLM call.

2.3 Corrective and Self-Reflective RAG

Several systems introduce retrieval evaluators that trigger corrective actions. CRAG proposes a corrective retrieval pipeline with confidence assessment and revision [7]. Self-RAG trains a model to retrieve, generate, and critique through self-reflection [8]. Antigravity adopts a lightweight, production-oriented variant: a confidence gate based on retrieval scores, followed by automatic query reformulation and iterative retrieval when confidence remains low.

2.4 Graph-Based Retrieval and Associative Memory

Knowledge graphs and graph traversal can expose multi-hop relations that dense similarity misses. Antigravity maintains a personal KG (entities, relations, temporal metadata) and uses Personalized PageRank (PPR) for associative recall [10]. This is especially important for relationship queries where the answer is a *path*, not a single chunk.

2.5 Belief Revision and Truth Maintenance

Personal memory is non-stationary: older facts become obsolete and may contradict new evidence. Antigravity's memory notes implement a lightweight truth maintenance mechanism inspired by classic Truth Maintenance Systems (TMS) [11] and the AGM belief revision framework [12]. The system detects supersession chains and can "unwind" them to reconstruct decision timelines.

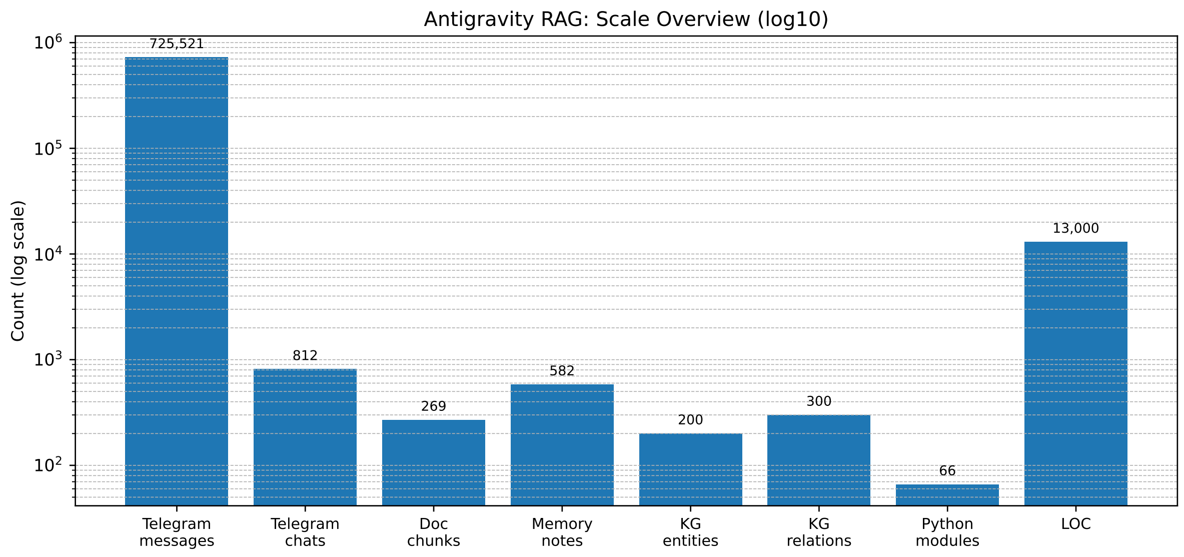

Figure 2. Scale overview (log scale). Telegram messages dominate raw volume; structured tables (notes, KG) provide a compressed, queryable representation. Counts reflect the February 2026 deployment snapshot.

Table 1. Primary source types and ingestion pipelines.

Source	Scale	Ingestion method
Telegram messages	725,521 msgs / 812 chats	Telethon sync → SQLite → Chat Digester → LanceDB
Daily logs	~60 files	Markdown chunking → LanceDB
People profiles	~20 files	Markdown chunking → LanceDB
Project docs	variable	3-level indexer (root index → per-project → code digests)
Blog posts	variable	Markdown/HTML chunking
Email	*planned*	Gmail read-only API ingestion

3 System Overview

Antigravity RAG is implemented as 50+ production modules (66+ modules including utilities/tests) totaling ~13 K+ lines of Python. It is served via a FastAPI daemon and supports retrieval (`/search`) and answering (`/answer`) endpoints.

3.1 Scale

Figure 2 summarizes the deployment snapshot (February 2026). Telegram dominates raw volume, while structured artifacts (notes, entities, relations, summaries) remain compact and query-efficient.

4 Data Sources and Ingestion

Antigravity ingests heterogeneous sources (Table 1). Ingestion outputs chunked documents with embeddings, extracted entities/relations, and atomic memory notes.

4.1 Chunking and Parent/Child Indexing

The ingestion pipeline implements multiple chunking strategies: section-aware Markdown splitting, sliding windows for chats, and a parent/child strategy where small "child" chunks are embedded for precise retrieval but larger "parent" contexts are returned to preserve coreference and narrative continuity.

4.2 Chat Digester for Noisy Conversational Data

The chat digester processes Telegram messages using tiered contact handling and temporal block summarization. It includes aggressive junk filtering (greetings, emoji-only, short logistics), crash-resilient checkpointing after each digest period, and streaming local inference to avoid timeouts on CPU.

4.3 Embedding Cache and Table-of-Contents Index

To reduce ingestion cost, Antigravity uses a persistent embedding cache keyed by (*text hash*, *model name*) to avoid re-embedding unchanged chunks. For iterative retrieval, a document-level table-of-contents (ToC) index stores per-section summaries so the system can assess document relevance without loading full content.

5 Query Processing

5.1 Seven-Intent Query Router

Every query is classified into one of seven intents: WHO_IS, DECISION, HOW_TO, RECENT, RELATIONSHIP, EVOLUTION, or GENERAL. Classification uses a near-zero-latency keyword stage (multilingual regex) and a centroid-embedding fallback that overrides GENERAL above a similarity threshold. Each intent compiles an explicit retrieval plan: ordered stages with per-stage limits, source filters, and parameter overrides (e.g., disabling time decay for EVOLUTION).

5.2 Coreference Resolution, Decomposition, and Iteration

For multi-turn interactions, the router optionally rewrites queries to resolve pronouns using a local CPU model. Complex queries can be decomposed into sub-queries and merged. When retrieval confidence is low, an iterative loop generates refined sub-queries and accumulates evidence for up to three rounds.

6 Retrieval Methods

Antigravity uses multiple retrieval mechanisms selected by intent.

6.1 Hybrid Vector + Full-Text Retrieval

The default retrieval combines dense vector search with BM25 full-text search and merges rankings with Reciprocal Rank Fusion (RRF) [5, 13]. Intent-aware source boosting favors stable profile/notes for WHO_IS and code-heavy sources for HOW_TO. For RECENT queries, optional exponential time decay penalizes older results:

$$s' = s \cdot \exp(-\lambda \cdot \Delta t), \tag{1}$$

where s is the base relevance score, λ is a tunable decay rate, and Δt is the age of the document in days.

6.2 Memory Notes and Timeline Unwinding

Atomic notes (FACT, DECISION, PROCEDURE, etc.) provide a high-signal representation of personal knowledge with provenance. For EVOLUTION queries, the system unwinds supersession chains to reconstruct belief/decision timelines (e.g., vector storage migrations from Pinecone to LanceDB).

6.3 Knowledge Graph Traversal and Personalized PageRank

Antigravity extracts entities and directed relations per chunk, storing temporal metadata and confidence scores. For associative recall, it runs Personalized PageRank seeded by query-matched entities [10], surfacing related entities and evidence that may be missed by similarity search alone.

6.4 Hierarchical and Community Summaries

For broad queries, Antigravity performs nightly clustering and summarization in a RAPTOR-like tree [6]. Entity communities are detected via graph clustering and summarized into compact representations for theme-level questions.

7 Post-Retrieval Processing

7.1 Confidence Gating and Reformulation

Antigravity computes a retrieval confidence score from top result scores, coverage, and score spread. If confidence drops below 0.5, the system triggers query reformulation and re-runs retrieval—inspired by CRAG [7]. Up to three reformulation rounds are permitted before falling back to the available evidence.

7.2 Cross-Encoder Reranking on CPU

Retrieved chunks are reranked with a multilingual cross-encoder (`BAAI/bge-reranker-v2-m3` [14]) that scores query–document pairs directly. This offers a favorable quality/latency tradeoff compared to LLM-based reranking on CPU, adding 0.5–2 s of latency for 30 chunks (Table 2).

7.3 Context Packing and Anti-Poisoning

The context packer allocates token budgets by intent (graph briefs vs. notes vs. raw evidence) and reorders chunks using a U-shaped "Gold Sandwich" strategy motivated by [9]: the most relevant chunks occupy the beginning and end of the context window where LLMs attend most reliably. All retrieved content is wrapped in CDATA inside XML with explicit untrusted-data markers and sanitization to mitigate prompt injection.

8 Caching and Memory Tiering

8.1 Two-Level Cache

Antigravity uses an in-memory exact-match cache (L0, hash-based) and a semantic cache (L1, vector similarity over query embeddings) with intent-aware TTL. Cache entries are invalidated on writes; semantic invalidation deletes nearby cached queries when new documents are ingested.

Table 2. Latency benchmarks on CPU-only VPS (8 vCPU AMD EPYC, 32 GB RAM, no GPU).

Operation	Latency
L0 cache hit	<1 ms
FTS search (BM25)	∼30 ms
Vector search (LanceDB)	∼50 ms
Embedding (warm, cached model)	∼800 ms
Full search pipeline (cache hit)	<100 ms
Cross-encoder rerank (30 chunks)	0.5–2 s
Full search pipeline (cache miss)	2–5 s
Embedding (cold start, model load)	5–15 s

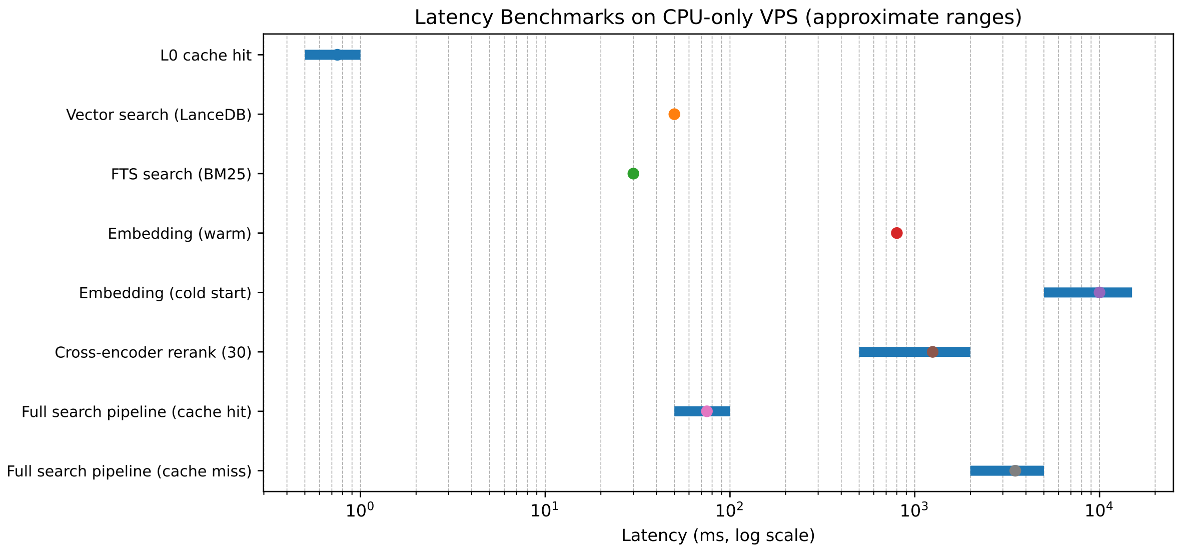

Figure 3. Latency ranges derived from Table 2 (log scale). Operations are sorted by typical latency. For "<" entries, the plot shows a short span near the upper bound for readability.

8.2 Hot/Warm/Cold Lifecycle and Rollups

Daily logs undergo automatic tiering: *hot* (0–3 days) kept verbatim, *warm* (4–30 days) compressed to summaries, and *cold* (30+ days) archived after extracting key facts. When an entity accumulates many micro-facts, rollups synthesize macro-facts and archive the originals to control growth.

9 Infrastructure and Performance

9.1 Cascading LLM Fallback

The LLM client routes requests through a cascading fallback: **Anthropic API** → **Qwen API** → **local Ollama (CPU-only)**. Timeouts, rate limits, and moderation blocks trigger automatic fallback. This architecture ensures availability even during provider outages while allowing local privacy-preserving execution as the last resort.

9.2 CPU-Only Benchmarks

Table 2 reports measured CPU-only latencies on the deployment VPS. Figure 3 visualizes these ranges on a log scale.

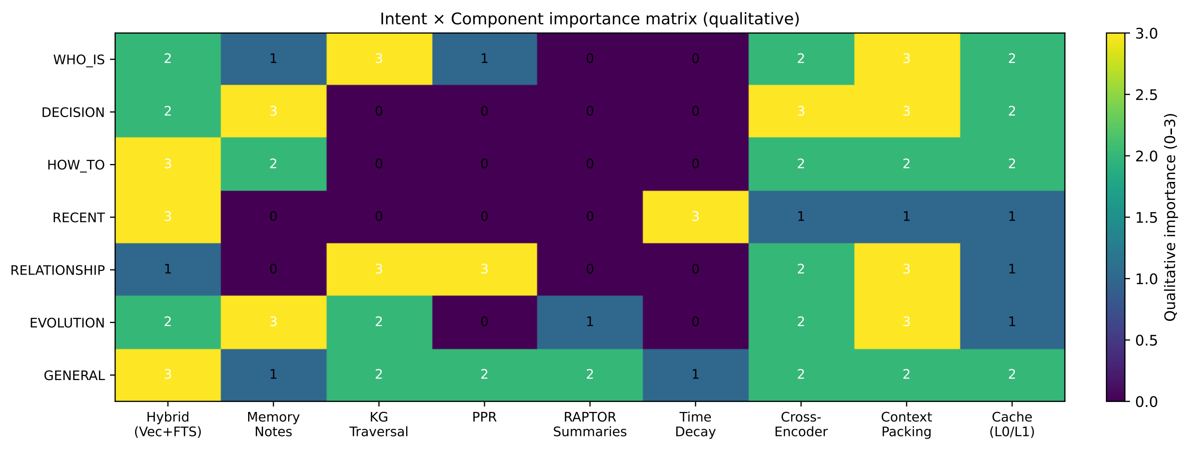

Figure 4. Intent × component importance matrix (qualitative, 0–3 scale). This figure reflects design-time routing and practical failure modes observed during development. It serves as an "ablation map" indicating which components are critical for each intent type.

9.3 Quality Evaluation

Antigravity includes a manual evaluation suite with curated query-answer pairs across all seven intents, measuring hit@5, MRR, and LLM-judged relevance. Automated evaluation is integrated via DeepEval [15] for faithfulness, relevancy, and contextual precision/recall.

10 Ablation and Intent-Wise Analysis

Because autobiographical retrieval mixes heterogeneous query types, we analyze the system at the level of *intents* and the components each intent depends on. Figure 4 summarizes qualitative importance (0–3) based on routing rules and observed failure modes during debugging. Table 3 provides expected impact per ablated component.

11 Ethics, Privacy, and Safety Considerations

Antigravity is a personal system that aggregates sensitive communications. This makes privacy and safety requirements first-class design concerns.

11.1 PII and Consent

Telegram chats, emails, and daily logs may contain personal data about the user and third parties. Operators should define retention and deletion policies, obtain consent where appropriate, and avoid ingesting content that cannot be ethically stored long-term (e.g., confidential third-party documents).

11.2 Cloud Fallback and Data Exposure

Although Antigravity can run fully locally, the cascading fallback may send prompts or retrieved context to third-party providers. We recommend: (i) local-first defaults; (ii) intent-based allow/deny rules for cloud usage (e.g., disallow WHO_IS for non-user persons); (iii) optional redaction of emails, phone numbers, addresses, and identifiers before remote calls; and (iv) provider-level logging controls when available.

Table 3. Component ablations: expected impact by intent and latency. Latency deltas use measured ranges when available; quality impacts are qualitative (High/Med/Low).

Ablated component	Most affected intents	Quality impact	Latency impact
Cross-encoder off	ALL (esp. GENERAL, HOW_TO)	Med–High (irrelevant context)	−0.5 to −2 s
PPR off (graph-only)	RELATIONSHIP, GENERAL	High (miss multi-hop)	Small
Belief revision off	EVOLUTION, DECISION	High (contradictions)	None
Time decay off	RECENT	High (stale answers)	Small
RAPTOR/summaries off	GENERAL (broad)	Med (no macro context)	Small
Cache off (L0/L1)	ALL (repeated)	Low (same evidence)	+seconds
Parent/child off	WHO_IS, GENERAL	Med (coreference fails)	Small

11.3 Prompt Injection and Data Poisoning

Because raw messages can contain malicious instructions, the system treats all retrieved text as untrusted. Mitigations include regex-based pre-flight screening, XML context packaging with CDATA, removal of dangerous tags, and explicit untrusted-data markers. These defenses reduce, but do not eliminate, risks from prompt injection and data poisoning.

11.4 Operational Security

We recommend encrypting disks, restricting API exposure (private network/VPN), authenticating all requests, and minimizing sensitive logs (e.g., avoid logging full retrieved chunks in plaintext). Backups should be encrypted and access-audited.

12 Lessons Learned

Empirically, we found:

1. **CPU-only inference is viable** for personal RAG at ∼10 tokens/s; generation, not retrieval, dominates end-to-end latency.
2. **Thread count matters:** using physical cores outperformed logical cores (hyperthreading) on the deployment CPU due to SMT contention.
3. **Incremental saves are essential** for long-running CPU-bound batch jobs such as the chat digester and RAPTOR build.
4. **Belief revision is critical** for preventing indefinite contradictions in personal memory—without supersession chains, stale facts persist indefinitely.
5. **Intent routing outperforms** "one pipeline fits all" retrieval.
6. **Cross-encoder reranking** is a practical CPU alternative to LLM-based scoring with favorable quality/latency tradeoffs.

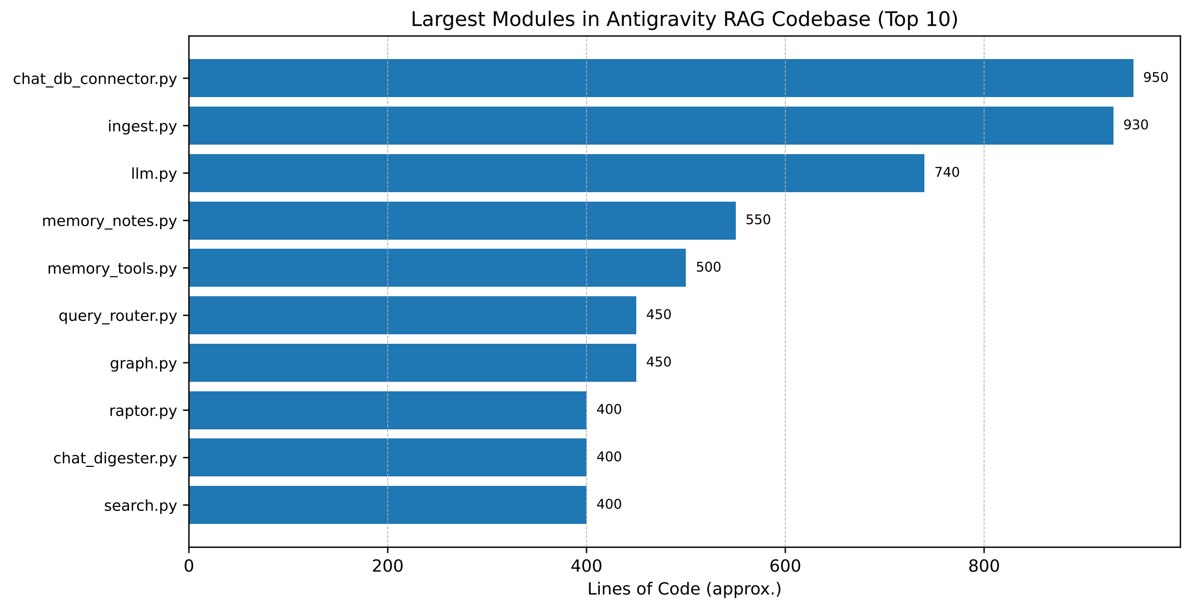

Figure 5. Largest modules (Top 10 by approximate lines of code). The system totals ∼13,000 LOC across 66+ Python modules.

7. **U-shaped context ordering** (Gold Sandwich) improves robustness to long-context degradation [9].
8. **Aggressive chat junk filtering** is necessary for high-quality digests.

13 Codebase Anatomy

Figure 5 highlights the largest modules by lines of code. The two largest—`chat_db_connector.py` (950 LOC) and `ingest.py` (930 LOC)—reflect the complexity of heterogeneous data source handling. The LLM client (`llm.py`, 740 LOC) encapsulates the cascading fallback, token budget management, and streaming support.

14 Limitations and Future Work

Current limitations. CPU-only throughput constraints limit ingestion speed for large batch jobs. The cross-encoder reranker has incomplete production deployment pending a dependency resolution. Email integration is planned but not yet implemented. Formal evaluation benchmarks specific to autobiographical retrieval do not yet exist; we rely on manually curated query–answer pairs.

Future work. Email ingestion via Gmail API; sliding-window embedding refresh after model upgrades; more agentic retrieval with tool-use loops; voice I/O integration for ambient capture; systematic epistemic sweeps for note staleness and inconsistency resolution; auto-tuning of intent router patterns via user feedback; and a formal evaluation framework for autobiographical RAG including annotated benchmark datasets.

15 Conclusion

Antigravity RAG demonstrates that a CPU-first, hybrid local/cloud architecture can support production-grade autobiographical retrieval over large personal corpora. The combination of intent-aware retrieval planning, belief-revision notes, knowledge-graph associative recall, hierarchical summaries, and robust post-retrieval controls provides a practical blueprint for lifelong personal memory systems. Running continuously on a single CPU-only VPS, the system ingests and retrieves from over 725,000 conversational messages while maintaining sub-5-second end-to-end latency on cache misses.

Prime Axiom. *"If it's not saved to disk, it doesn't exist."*

References

[1] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. arXiv:2005.11401.

[2] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. REALM: Retrieval-Augmented Language Model Pre-Training. In *International Conference on Machine Learning (ICML)*, 2020. arXiv:2002.08909.

[3] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of EMNLP*, 2020. arXiv:2004.04906.

[4] Omar Khattab and Matei Zaharia. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In *Proceedings of SIGIR*, 2020. arXiv:2004.12832.

[5] Gordon V. Cormack, Charles L. A. Clarke, and Stefan Büttcher. Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods. In *Proceedings of SIGIR*, 2009.

[6] Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval. *arXiv preprint* arXiv:2401.18059, 2024.

[7] Shi-Qi Yan, Jia-Chen Gu, Yun Zhu, and Zhen-Hua Ling. Corrective Retrieval Augmented Generation. *arXiv preprint* arXiv:2401.15884, 2024.

[8] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. *arXiv preprint* arXiv:2310.11511, 2023.

[9] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics (TACL)*, 2023. arXiv:2307.03172.

[10] Glen Jeh and Jennifer Widom. Scaling Personalized Web Search. In *Proceedings of the 12th International World Wide Web Conference (WWW)*, pages 271–279, 2003.

[11] Jon Doyle. A Truth Maintenance System. *Artificial Intelligence*, 12(3):231–272, 1979.

[12] Carlos Alchourrón, Peter Gärdenfors, and David Makinson. On the Logic of Theory Change: Partial Meet Contraction and Revision Functions. *Journal of Symbolic Logic*, 50(2):510–530, 1985.

[13] LanceDB. Hybrid Search: Combining BM25 and Semantic Search for Better Results with LanceDB. Blog post, December 2023. https://lancedb.com/blog/hybrid-search-combining-bm25-and-semantic-search-for-better-results-with-lan-1358038fe7e Accessed 2026-02-24.

[14] BAAI. BGE-Reranker-v2 Documentation: `BAAI/bge-reranker-v2-m3`. https://bge-model.com/bge/bge_reranker_v2.html. Accessed 2026-02-24.

[15] Confident AI. DeepEval: The Open-Source LLM Evaluation Framework. https://github.com/confident-ai/deepeval. Accessed 2026-02-24.