

The Zain Alhassani Algorithm

A Disk-First Memory Architecture for Semantic Search on Low-RAM Hardware

Zain Al-Abdeen Alhassani

Jarvice Digital · Baghdad, Iraq

zaioni2015@gmail.com · zikodev.com

Version	1.0
Date	26 October 2026
System	Ziko Brain, knowledge memory of the Jarvice AI platform
Status	Implemented and running in production since 24 October 2026
Copyright	© 2026 Zain Al-Abdeen Alhassani. All rights reserved.

Abstract

AI assistants that learn continuously store each learned fact as a high-dimensional vector and retrieve facts by semantic similarity. The common design keeps every vector in main memory, so memory use grows linearly with knowledge and quickly exceeds the capacity of small servers. This paper presents the Zain Alhassani Algorithm, a disk-first memory architecture in which all facts and vectors reside on persistent storage and main memory holds only a compact *map* of cluster centres and storage offsets. A query consults the map, reads only the nearest clusters from disk through memory mapping, scores them with 8-bit compressed vectors and re-ranks the best candidates with their exact vectors. Facts added after the last map build are searched directly, so new knowledge is available immediately. In a controlled test with 60,000 facts, the algorithm reduced the memory of the search process from 467 MB to 82 MB and the search time from up to 30 ms to about 2 ms, while returning the same best result as exhaustive search in 96–100% of queries. On the production system it returned identical results to the previous design and cut search time from 16.5 ms to 6.8 ms.

Keywords: semantic search, vector retrieval, approximate nearest neighbour, memory mapping, quantization, low-resource AI, knowledge base

1. Introduction

Ziko Brain is the self-learning knowledge memory of the Jarvice AI platform. It learns from public sources around the clock, indexes the platform's own source code, and supplies relevant facts to the language model before each answer. Every fact is embedded as a 768-dimensional vector. Retrieval finds the stored vectors closest to the embedded question.

The original design loaded all vectors into main memory at start-up and compared each question with every vector. This is simple and exact, but its memory cost is proportional to the number of facts: about 3 KB per fact in 32-bit form, or roughly 3 GB per million facts. The production server has 7.8 GB of RAM shared with a web application, background workers and locally hosted language models. Memory, not storage, therefore set the ceiling on how much the system could learn.

The central idea of this work is to invert the usual arrangement: **knowledge lives on the disk, and memory holds only the map needed to find it**. This paper describes the architecture (Section 3), its implementation (Section 4), its measured performance (Section 5), and its limits (Section 6).

2. Background and related work

Partitioning vectors by their nearest cluster centre and searching only the closest partitions is the inverted-file approach introduced for image retrieval [1] and widely used in vector search libraries such as FAISS [3]. Compressing vectors to reduce memory and bandwidth is well established, from scalar quantization to product quantization [2]. Disk-resident vector indexes have been studied for billion-scale search, notably DiskANN [4] and SPANN [5], which keep compact navigation data in memory and full data on solid-state storage.

The present work applies these principles to a different setting: a continuously learning assistant on a small general-purpose server, where the index must accept new facts at any time, rebuild without interrupting service, coexist with other memory-hungry processes, and fall back safely to exact search. Its contribution is the combination and engineering of these elements for that setting, as listed in Section 3.6.

3. The algorithm

3.1 Storage layout

Tier	Contents	Size per fact
Main memory: the map	Cluster centres, cluster start offsets, fact identifiers, per-vector scale factors	about 12 bytes, plus the centres
Disk: vector file	All vectors in 8-bit form, stored contiguously and grouped by cluster	768 bytes
Disk: database	Fact text, source, topic and exact 32-bit vector	text + 3,072 bytes

Table 1. The three storage tiers. Only the first tier must reside in main memory.

3.2 Building the map

Let N be the number of facts. A random sample of up to 20,000 normalised vectors is clustered with spherical k-means into $C = \min(1024, \max(16, 2\sqrt{N}))$ centres. All vectors are then streamed from the database in batches. Each vector is assigned to its nearest centre and compressed to 8 bits with its own scale factor $s = \max|v| / 127$. The compressed vectors are written grouped by cluster, so that each cluster occupies one contiguous region of the file. The map records where each region starts. The build never holds more than one batch of full vectors in memory, and new files replace old ones atomically.

3.3 Answering a query

Algorithm 1. Query procedure

Input: normalised query vector q , number of results k , probes P (default 32)

1. Score q against every cluster centre held in the map.
2. Select the P clusters with the highest scores.
3. For each selected cluster, read its contiguous region from the memory-mapped vector file and compute approximate scores $(\text{int8 vector} \cdot q) \times \text{scale}$.
4. Keep the best $\max(4k, 20)$ candidates across all selected clusters.
5. Fetch those candidates, and every fact added after the last build (the delta), from the database with their exact 32-bit vectors.
6. Re-score all fetched facts exactly and return the top k .

Only the selected regions of the vector file are read, and the operating system caches frequently used regions automatically. The exact re-scoring in step 6 removes most of the error introduced by compression.

3.4 Incremental learning

New facts are written to the database as before. Facts whose identifier exceeds the highest identifier included in the last build form the *delta*, which every query searches exactly (step 5). New knowledge is therefore retrievable the moment it is stored, with no need to rebuild the map first.

3.5 Maintenance and safety

The map is rebuilt every hour by a scheduled service at low CPU and I/O priority, under a 600 MB memory cap, and the new files replace the old ones atomically. The search layer detects new files and reloads the map without a restart. If the index files are missing or unreadable, the system falls back automatically to the original exact search, so correctness never depends on the index. Small tables (fewer than 500 facts) use exact search directly.

3.6 Contributions

(1) A disk-first memory architecture for continuously learning assistants, in which main memory holds only a navigation map. (2) A delta mechanism that makes new knowledge searchable immediately, without a rebuild. (3) A maintenance design (hourly low-priority rebuilds, atomic replacement, hot reload, automatic fallback) suited to small shared servers. (4) A production deployment and measurements on real and synthetic data.

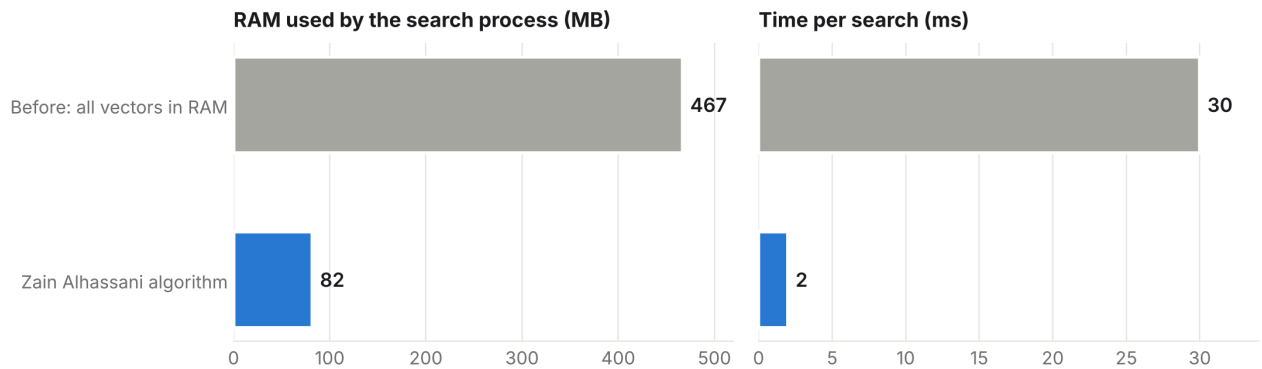
4. Implementation

The reference implementation is a Python module (`diskbrain.py`, about 170 lines) that uses NumPy and SQLite, with no external search library. It is integrated into Ziko Brain's memory layer (`memory.py`) and is used both for learned facts and for the instant-answer cache. Vectors come from the `nomic-embed-text` embedding model (768 dimensions). The deployment includes a backup of the original memory layer, a side-by-side accuracy test before activation, and a one-command rollback.

5. Evaluation

5.1 Setup

Controlled test: 60,000 synthetic facts generated around 400 topic centres with noise, 768-dimensional normalised vectors, 50 queries formed by perturbing stored vectors. Both designs ran on the same machine, and the reference for accuracy was exhaustive exact search. **Production test:** the live Ziko Brain database (809 facts), 30 queries formed from stored facts, comparing the new search with the previous one.



Test: 60,000 facts, 768-dimension vectors, 50 searches, same machine. Lower is better.

Figure 1. Memory of the search process and time per search, controlled test with 60,000 facts. Lower is better.

Measure	Previous design	Zain Alhassani Algorithm
Controlled test (60,000 facts)		
Memory of the search process	420–467 MB	82 MB
Time per search	19–30 ms	about 2 ms
Same best result as exact search	100% (reference)	96–100%
Vector file on disk	—	46 MB
Production (809 facts, 24 October 2026)		
Time per search	16.5 ms	6.8 ms
Memory of the search process	55 MB	50 MB
Same best result as previous design	—	100%

Table 2. Measured results.

Clusters probed (P)	Same best result	Same top-5 results	Time per search
16	92%	88%	1.3 ms
32 (default)	96%	95%	2.0 ms
48	98%	98%	2.2 ms

Table 3. Accuracy and speed against the number of clusters probed, controlled test.

Facts	Previous design: vectors in memory	Map in memory	Vector file on disk
60,000	184 MB	2 MB	46 MB
1,000,000	3.1 GB	15 MB	768 MB
10,000,000	30.7 GB	123 MB	7.7 GB

Table 4. Projected vector storage by scale, calculated from the design (not measured). Text storage and interpreter overhead are excluded.

6. Discussion and limitations

Approximation. With the default setting, about 4% of queries return a different best fact. These are typically facts of almost equal similarity. Accuracy can be raised by probing more clusters at a small cost in time (Table 3).

Scope. The method suits knowledge retrieval, where each query needs only a few related items. It does not make it practical to run large language models from disk, because a dense model reads all of its weights for every generated token, and on the production server's storage (about 33 MB/s sequential read) that cost is prohibitive.

Evidence. The large-scale results use synthetic vectors, and the production database is still small. The measurements should be repeated as the knowledge base grows, and on other hardware.

7. Conclusion

By keeping knowledge on disk and only a navigation map in memory, the Zain Alhassani Algorithm removes main memory as the limit on how much a self-learning assistant can know. In tests it cut search memory by about a factor of five and search time by about a factor of ten, at 96–100% agreement with exact search. It now runs in production as the memory of Ziko Brain on a server with 7.8 GB of RAM, where it allows the knowledge base to grow by orders of magnitude without a corresponding growth in memory.

Acknowledgements

The reference implementation and the evaluation scripts were developed with the assistance of an AI coding assistant, under the author's direction and specification.

References

1. J. Sivic and A. Zisserman. "Video Google: A text retrieval approach to object matching in videos." *Proc. IEEE ICCV*, 2003.
2. H. Jégou, M. Douze and C. Schmid. "Product quantization for nearest neighbor search." *IEEE Trans. Pattern Analysis and Machine Intelligence*, 33(1), 2011.
3. J. Johnson, M. Douze and H. Jégou. "Billion-scale similarity search with GPUs." *IEEE Trans. Big Data*, 7(3), 2021 (FAISS).
4. S. J. Subramanya, F. Devvrit, R. Kadekodi, R. Krishnaswamy and H. V. Simhadri. "DiskANN: Fast accurate billion-point nearest neighbor search on a single node." *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.
5. Q. Chen et al. "SPANN: Highly-efficient billion-scale approximate nearest neighbor search." *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Appendix A. Parameters

Parameter	Value	Purpose
Vector dimension	768	Set by the embedding model
Clusters C	$\min(1024, \max(16, 2\sqrt{N}))$	Size of the map
Training sample	up to 20,000 vectors	Bounds the build cost
Probes P	32 (setting ZK_DISK_NPROBE)	Accuracy against speed
Candidates re-ranked	$\max(4k, 20)$	Exact re-scoring
Compression	8-bit, one scale per vector	4× smaller vectors
Rebuild	hourly, low priority, 600 MB cap	Folds new facts into the map
Exact-search threshold	fewer than 500 facts	Small tables need no index