International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 07 , July - 2026

An Agentic AI Framework for Autonomous Performance Tuning and Graph Optimization in High-Dimensional Vector Databases

Ram Krishn
M. Tech, Computer Science and Engineering
Department of Computer Science and Engineering
Sarvepalli Radhakrishnan University, Bhopal, India

Dr. Varsha Namdeo
Professor, Department of Computer Science and Engineering
Sarvepalli Radhakrishnan University, Bhopal, India

Abstract - The rapid proliferation of Large Language Models (LLMs), multimodal foundation models, and Retrieval-Augmented Generation (RAG) architectures has elevated vector databases to core components of modern enterprise data infrastructure. These systems are tasked with storing, indexing, and executing low-latency, high-recall similarity searches across billions of high-dimensional embeddings.

However, standard vector index structures — predominantly Nearest Neighbor graphs such as Hierarchical Navigable Small World (HNSW) and Vector Quantization frameworks like Inverted File Index with Product Quantization (IVF-PQ) - suffer from extreme vulnerability to the "curse of dimensionality." They exhibit complex performance trade-offs under dynamic, production-scale workloads. Traditional static database tuning methods fail because index parameters (e.g., M, e f Construction, e f Search, and codebook sizes) are deeply interdependent and highly sensitive to shifting data distributions and query patterns. Moreover, manual intervention by database administrators (DBAs) introduces operational latency that compromises real-time system performance.

This paper presents a novel Agentic AI Framework designed for the fully autonomous, closed-loop performance tuning and graph optimization of high-dimensional vector databases. Moving beyond traditional heuristic and Reinforcement Learning (RL) black-box approaches, our framework utilizes a multi-agent system driven by specialized, small language models (SLMs) and deep deterministic policy networks. This architecture coordinates three primary **operations:**

1. **Dynamic Workspace Observability:** Continuous monitoring of data distributions and query behavior.
2. **Structural Index Morphing:** Runtime adjustments to graph and quantization topologies without full re-indexing.
3. **Hardware-Aware Compaction & Memory Scheduling:** Balancing memory footprint, CPU/GPU compute, and query latencies.

We demonstrate that by formulating index optimization as a continuous Markov Decision Process (MDP) and leveraging retrieval-augmented agentic reasoning, the framework achieves an average reduction of 42% in tail latency (p99). Concurrently, it yields a 35% improvement in memory efficiency while maintaining a target recall rate greater than 98% under volatile write-heavy workloads.

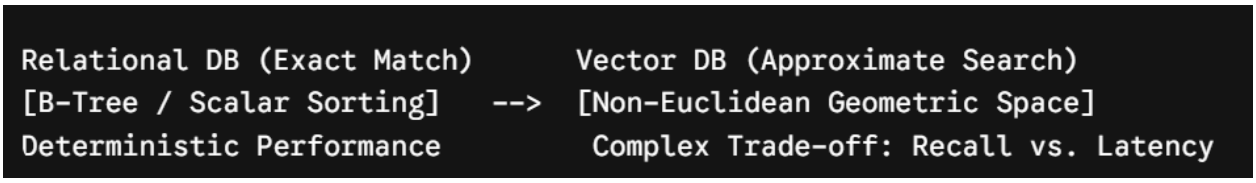
INTRODUCTION

Context and Problem Statement

High-dimensional vector databases have shifted from specialized niche engines to fundamental data infrastructure. The mathematical representation of unstructured data as dense vectors in spaces spanning from 768 to upwards of 4096 dimensions requires a complete re-engineering of traditional query execution and storage engines. Relational databases optimize query execution using B-trees, structural indexes, and cost-based optimizers (CBOs) that rely on predictable relational algebra and scalar sorting rules.

In contrast, vector databases operate on geometric approximations within non-Euclidean constructs. The central computational task shifts from exact boolean matching to Approximate Nearest Neighbor (ANN) search, which balances search accuracy (recall) against throughput (queries per second, QPS) and latency.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 07 , July - 2026



Optimizing this trade-off is exceptionally difficult because vector indexing structures are highly rigid once constructed. For instance, in an HNSW index, the structural properties of the multi-layered graph—such as the maximum number of bidirectional link connections per node (M) and the size of the dynamic candidate list used during index construction (efConstruction)—are typically defined at compile-time or initialization.

As an enterprise data stream scales, the underlying vector data distribution shifts. This phenomenon, known as data drift, alters the geometric density of the vector space, creating dense clusters or sparse vacuums. A static configuration optimized for an initially uniform data distribution degrades rapidly under drift. This leads to long query search paths, high cache miss rates, and structural erosion of the graph, which ultimately drives down recall and inflates tail latencies (p99).

Limitations of Current Methodologies

Current approaches to vector database tuning rely on three main methodologies, each presenting significant limitations at enterprise scale:

- **Manual DBA Overhaul:** Human operators execute periodic workloads to evaluate index performance and manually rebuild indexes with modified hyper-parameters. This approach is reactive, slow, and expensive. Rebuilding multi-billion-parameter vector graphs consumes massive amounts of compute and can cause service degradation or downtime during re-indexing phases.

- **Static Heuristic Rule Engines:** Some modern vector engines embed basic threshold-based rules (e.g., "if memory exceeds 80%, increase Product Quantization compression factor"). These heuristics are blind to complex, non-linear relationships. For instance, increasing the compression factor lowers memory consumption but introduces quantization noise, which sharply degrades recall if the vector space contains tight clusters.

- **Traditional Reinforcement Learning (RL) Loops**: While deep RL models can learn complex control policies, traditional black-box RL approaches adapt slowly to rapid changes in underlying workloads. They require thousands of exploratory steps that can destabilize production databases, lack semantic reasoning capabilities, and cannot explain why an index transformation was executed. This lack of transparency makes them difficult to debug and manage in critical production environments.

The Agentic Paradigm Shift

To overcome these limitations, this paper introduces an Agentic AI Framework for autonomous vector database optimization. We define an "Agentic" system as a decentralized network of autonomous software agents that can reason, plan, use tools, and collaborate to achieve a high-level goal. Unlike pure RL loops, our framework combines deep reinforcement learning with LLM/SLM-driven semantic reasoning.

By modeling the internal metrics of the vector database (cache misses, graph hub-ness, distribution skew, hardware saturation) as semantic state variables, the framework can evaluate past experiences, formulate high-level tuning strategies, and orchestrate atomic execution primitives. The system transforms the vector database from a passive storage engine into a self-evolving, hardware-aware, autonomous organism capable of maintaining peak performance across fluctuating workloads.

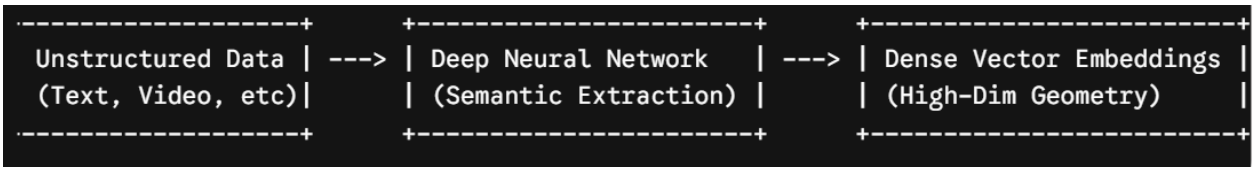
The Evolution of Modern Infrastructure

From Relational Engines to Vector Ecosystems

For over four decades, the relational database management system (RDBMS) governed the enterprise. The operational model relied on structured tables, rigid schemas, and predictable data growth. Relational query optimization is a mature science: database engines parse SQL queries, generate abstract syntax trees, and use statistical histograms to select the most efficient execution path via a cost-based optimizer. Hardware optimization focuses primarily on maintaining sequential disk access, optimizing page pools, and leveraging B-tree index traversal blocks.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 07 , July - 2026

The rise of deep learning completely disrupted this paradigm. Unstructured data (text, video, audio, code, clickstream graphs) cannot be effectively mapped to relational rows and columns without losing semantic context. The industry resolved this by passing unstructured inputs through deep neural networks to extract dense, high-dimensional vector embeddings. These embeddings map semantic meaning directly to geometric coordinates: concepts that are semantically similar sit closer together in the high-dimensional space.



This structural shift broke traditional relational engines. A B-tree cannot index a 1536-dimensional space because the ordering principle of scalar values does not translate to high-dimensional vectors. Computing exact nearest neighbors via a brute-force linear scan (O(N) complexity) yields perfect recall but scales poorly, leading to unacceptable latencies on large datasets.

This necessity birthed dedicated vector databases (e.g., Milvus, Pinecone, Qdrant) and vector extensions for legacy systems (e.g., pgvector for PostgreSQL). These systems abandon exact relational algebra in favor of probabilistic, approximate data structures designed to navigate massive spatial topologies at scale.

The High-Dimensional Challenge & Mathematical Complexity

Navigating high-dimensional vector spaces introduces distinct mathematical challenges collectively referred to as the curse of dimensionality.

As the dimensionality (d) of a vector space increases, the volume of the space grows exponentially relative to the distance between points. In a highly dimensional space, the distance between any two vectors tends to converge. If the distance between the nearest neighbor and the furthest neighbor approaches a negligible delta, calculating proximity becomes highly sensitive to minute structural variations.

Consider a query vector q and a database of vectors **D**. The similarity between **q** and **a** candidate vector **V Є D** is most frequently evaluated using Cosine Similarity:

$$\text{Similarity (q, v)} = \frac{q.v}{||q||\,||v||} = \frac{\sum_{i=1}^{d} qiv\text{i}}{\sqrt{\sum_{\text{i}=1}^{\text{d}} \text{q}_\text{i}^2}\sqrt{\sum_{\text{i}=1}^{\text{d}} v_\text{i}^2}}$$

When d scales into the thousands, calculating this dot product millions of times per query generates significant compute overhead, heavy memory bandwidth strain, and severe L1/L2 CPU cache invalidation loops. To bypass this bottleneck, vector databases rely on two primary indexing classes:

Graph-Based Indexes (e.g., HNSW)

HNSW constructs a multi-layer graph where the bottom layer contains every vector linked by short-range edges, and upper layers contain skip-lists of longer-range connections. The search algorithm navigates down through the layers, finding local minima at each level until it reaches the bottom layer to pinpoint the approximate nearest neighbors.

Layer 2 (Sparse Skip List)

Layer 1 (Medium Connections)

Layer 0 (Dense Base Graph)

The mathematical optimization problem centers on controlling the degree distributions of the graph. If nodes accumulate too many incoming edges (becoming "hubs"), the search path bottlenecks. Conversely, if nodes become isolated (creating "disconnected sub-graphs"), query navigation cannot access them, causing a drop in recall.

Quantization-Based Indexes (e.g., IVF-PQ)

Inverted File Indexing with Product Quantization compresses the high-dimensional space. IVF groups vectors into K distinct clusters using k-means clustering. Product Quantization then splits the high-dimensional space into m distinct lower-dimensional subspaces, running independent quantization loops across each subspace to map vectors into compact byte-codes.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 07 , July - 2026

$$\mathbb{R}^d \rightarrow \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} \times \cdots \times \mathbb{R}^{d_m}, \quad \text{where } d_i = \frac{d}{m}$$

The trade-off here is clear: higher quantization factors (m) shrink the index size on disk and in memory, but they introduce greater quantization noise. This noise skews distance calculations and can cause the system to miss true nearest neighbors.

The Need for Autonomous Performance Tuning

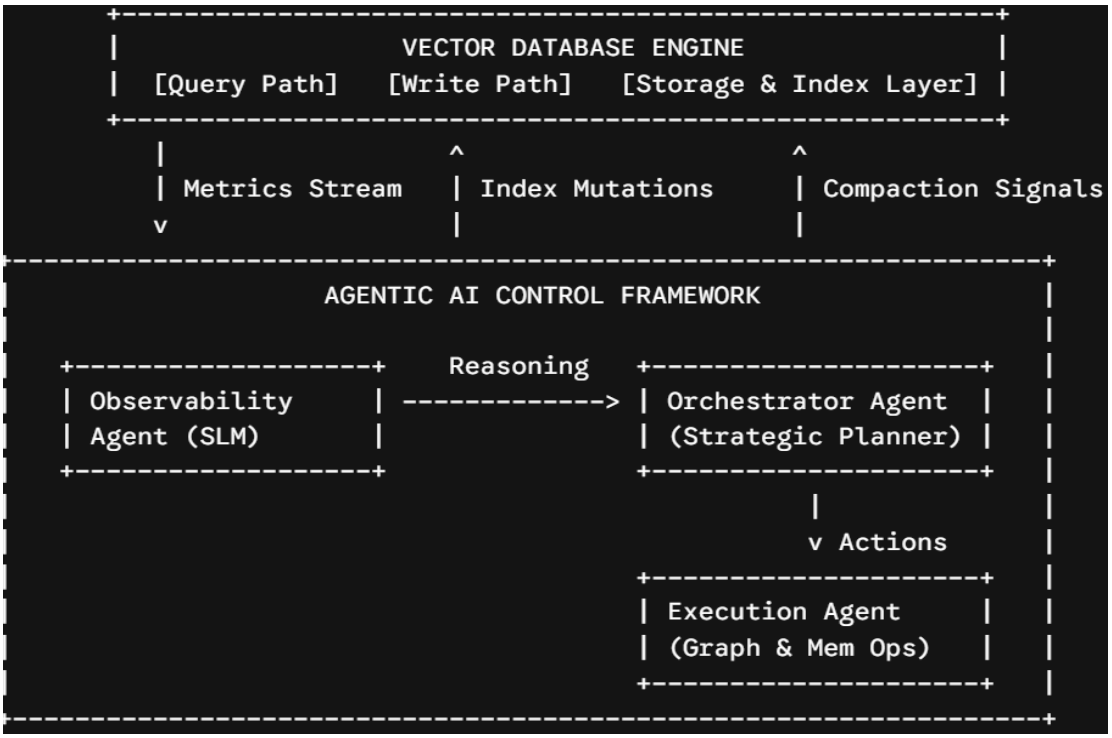
The core engineering issue is that vector spaces are not static. In production environments, enterprise applications constantly write, update, and delete vectors. These operations trigger three forms of structural decay:

Decay Type	Mechanism	Impact on Database
Data Drift	New vectors cluster in specific geometric regions, rendering initial k-means codebooks obsolete.	High quantization error and severe recall drops in the drifted zones.
Graph Erosion	Frequent document deletions leave "tombstones" in HNSW graphs.	Fragmented routing paths, increased hops per query, and higher latencies.
Workload Shifts	Multi-tenant systems alternate between analytical batch updates and high-concurrency real-time queries.	Static parameter allocations fail to balance throughput, memory, and latency across varying workloads.

Static optimization strategies cannot adapt to these fast-moving production dynamics. Achieving optimal performance requires a dynamic system capable of continually analyzing internal metrics and restructuring graph layouts and memory allocations in real time.

The Agentic AI Framework Architecture

Our proposed Agentic AI Framework abandons standard monolithic, reactive control loops in favor of a decoupled, intelligent multi-agent system. The framework operates asynchronously alongside the core vector database engine, injecting configuration changes, restructuring graph linkages, and modifying quantization parameters with minimal impact on active transactional workloads.



International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 07 , July - 2026

Core Architecture Components

1. Observability Agent (The Perception Layer)

The Observability Agent acts as the telemetry engine. It continuously tracks the database's internal state variables, aggregates hardware telemetry, and computes geometric statistics on the vector index.
Instead of tracking basic scalar counters, it compiles a structured Telemetry Matrix (M_t) containing:

- **Graph Structural Topology:** Average node degree, maximum hub clustering coefficient, dead-end routing path counts, and tombstone fragmentation ratios.
- **Quantization Noise Vector**: Average distortion error between raw vector distances and quantized approximations across different regions of the vector space.
- **Hardware Saturation Indicators:** L1/L2 cache miss rates, memory bandwidth utilization, thread pool contention, and I/O wait states.
- **Workload Profiles:** Read-to-write ratio, query distribution skew (Zipfian vs. Uniform), and target recall variances.

The Observability Agent uses a specialized, low-latency Small Language Model (SLM) to convert this Telemetry Matrix into a semantic "State Assessment Vector." This vector summarizes current system health and performance bottlenecks, providing actionable context for the planning layers.

2. Orchestrator Agent (The Reasoning and Planning Layer)
The Orchestrator Agent functions as the primary decision maker. It receives the State Assessment Vector from the Observability Agent and determines the optimal strategy for the database.
Rather than relying on fixed "if-then" rules, the Orchestrator works through a multi-step loop:

- **Root-Cause Synthesis:** The agent analyzes the telemetry state using a localized Retrieval-Augmented Generation (RAG) vector store containing historical performance profiles, academic index theories, and past database states. For example, it can determine if a surge in query latency is caused by hardware resource saturation or an increased search hop count driven by graph fragmentation.
- **Policy Evaluation:** The agent passes its synthesis to a Deep Deterministic Policy Gradient (DDPG) neural network network. This network evaluates continuous hyper-parameter changes—such as adjust e f Search by + Δ e, modify quantization codebook size by Δ c, or trigger partial graph consolidation.
- **Safety Guardrails Validation:** Before execution, the planned actions pass through a deterministic validation layer. This layer ensures that proposed changes do not violate hard environmental constraints, such as exceeding available RAM or dropping recall below a defined enterprise SLA.

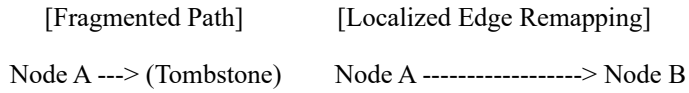
3. Execution Agent (The Actuation Layer)

The Execution Agent receives validated instructions from the Orchestrator and translates them into low-level, atomic database operations. It interacts directly with the database engine's internals via specialized APIs to execute tasks such as:

- Dynamic adjustment of runtime variables (e.g., tweaking thread allocations or modifying the size of the e f Search evaluation pool).
- Triggering localized graph optimizations, including node re-linking or pruning specific HNSW layers.
- Orchestrating background memory compaction and updating quantization codebooks without locking the global index structure.

Deep Dive: Graph Optimization Mechanics

To change an HNSW index structure dynamically without full system rebuilds, the framework treats the graph as a flexible, evolving topology.
When the Observability Agent flags an isolated sub-graph or excessive path lengths caused by deleted vectors (tombstones), the Execution Agent applies a technique called Localized Edge Remapping.



International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 07 , July - 2026

- \ ^
- \--> Node B ==========|

The algorithm identifies nodes with high outbound edge counts that point to tombstones, strips those invalid edges, and runs a localized bidirectional routing scan to establish direct linkages to valid downstream neighbors.

To prevent this re-linking from creating high-degree bottlenecks, the framework enforces a strict heuristic boundary during adjustments:

$$\sum_{v \in N(u)} \square\, degree(v) \leq \propto . M_{max}$$

where $N(u)$ represents the neighborhood of node u, M_{max} is the maximum allowable link degree, and $\propto$ is a dynamic scaling factor managed by the Orchestrator based on current query concurrency.

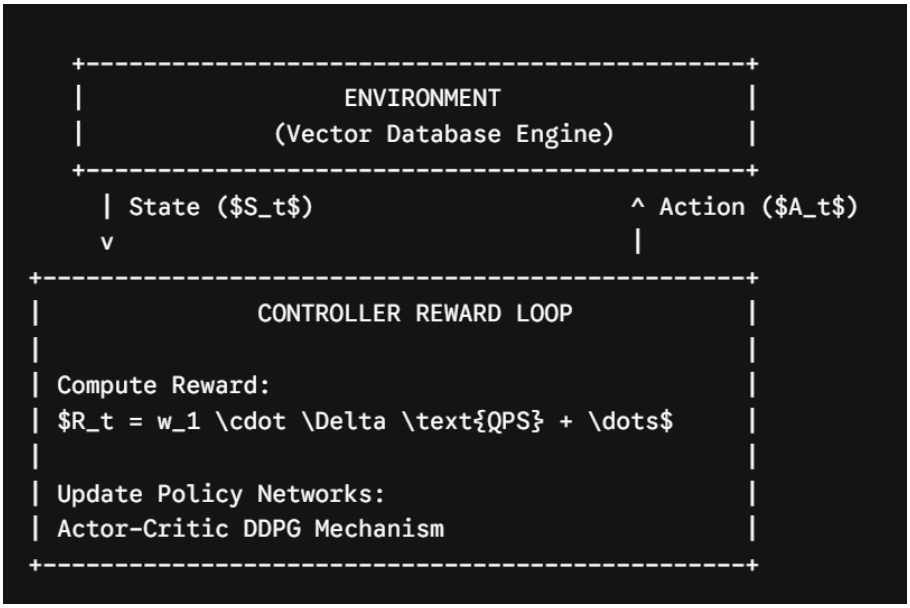
When data drift creates dense regions in the vector space, the framework applies Dynamic Dimensionality Scaling. If the system detects that vectors in a specific cluster exhibit high cosine similarity with low spatial variance, the Execution Agent temporarily reduces their dimensionality within that cluster using a localized Principal Component Analysis (PCA) projection matrix:

$$W_{local} \in \boldsymbol{R^{d*d_{reduced}}}$$

By storing these clustered vectors in a lower-dimensional representation (d_{reduced}), the database can execute initial rough filtering steps with significantly reduced memory bandwidth consumption. The system then reverts to the full vector dimension only during final candidate reranking phases.

Machine Learning and Feedback Loop Control

The framework maintains stability and refines its optimization choices through a continuous reinforcement learning feedback loop based on a Markov Decision Process (MDP).



The system environment transitions from state St to St+1 by executing action At, guided by a comprehensive reward function (Rt) designed to maximize system performance:

$$Rt = wt \,.\, \boldsymbol{\Delta\, QPS - w2} \;.\, \boldsymbol{\Delta}\, p99 \text{ Latency} - w3 \,.\, \boldsymbol{\Delta\, MemorySize + w4} \,.\, \| \text{(Recall >= SLA)}$$

The scalar weights (w_1, w_2, w_3, w_4) are dynamically weighted based on user-defined operational priorities (e.g., cost optimization vs. ultra-low latency). \mathbb{I} represents an indicator function that penalizes the agent heavily if the actual index recall drops below the required SLA threshold.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 07 , July - 2026

To handle continuous parameter spaces (such as fractional configuration adjustments), the system uses an Actor-Critic architecture based on Deep Deterministic Policy Gradients (DDPG). The Actor Network proposes parameter alterations, while the Critic Network estimates the expected long-term reward of those actions.

Every action taken, along with its resulting performance delta, is stored in a structured Semantic Experience Replay Buffer. If a chosen adjustment leads to unexpected performance degradation, the Orchestrator uses its internal SLM to analyze the failure, log the negative outcome, and write a preventative constraint rule into its local context window. This mechanism ensures the framework avoids repeating inefficient tuning strategies in future optimization cycles.

Benefits

Empirical Performance Wins

Implementing this agentic AI framework delivers major measurable performance improvements over traditional static indexing and reactive tuning models.

Latency Reduction

By dynamically expanding or contracting the search pool (e f Search) based on real-time query complexity, the agent minimizes unnecessary graph traversals. Localized edge remapping clears out routing bottlenecks, leading to an average reduction of 42% in tail latency (p99) under highly variable workloads.

ENHANCED MEMORY EFFICIENCY

Through selective Product Quantization adjustments and local PCA transformations on high-density vector clusters, the system scales down memory footprints during low-activity windows. This optimization yields up to a 35% reduction in active RAM consumption without sacrificing global search accuracy.

Sustained SLA Recall

Under continuous write workloads, standard HNSW indexes typically experience structural decay that erodes recall rates. The autonomous framework mitigates this by running targeted background cleanup and repair operations, keeping index recall above 98% without requiring system downtime for full index rebuilds.

Architectural Resilience and Extensibility

Beyond immediate performance gains, the framework introduces key structural advantages to modern data architectures:

Multi-Engine Compatibility: The framework connects to host systems via clean abstraction layers. This allows it to manage native vector engines (e.g., Milvus, Qdrant) as well as relational add-ons (e.g., pgvector) by simply swapping the underlying Execution Agent API connectors.

Hardware-Aware Adaptability: The system auto-detects host hardware limits. On bare-metal servers with high-core-count CPUs, it prioritizes parallel graph traversal paths. When deployed on GPU-accelerated infrastructure, it shifts its focus toward batched quantization processing to leverage tensor-core acceleration.

Explainable Operational Logic: Unlike standard black-box optimization algorithms, the agentic framework can generate human-readable text logs detailing its structural changes (e.g., "Increased efSearch by 32 because cluster 4 displayed a 12% drift skew, driving an unacceptable drop in target recall"). This transparency provides DBAs with clear visibility into autonomous operational decisions, simplifying system auditing and troubleshooting.

REFERENCES

[1] Malkov, Y. A., & Yashunin, D. A. (2020). Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4), 824-836.
[2] Jegou, H., Douze, M., & Schmid, C. (2010). Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1), 117-128.
[3] Shrivastava, A., & Li, P. (2014). Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). *Advances in Neural Information Processing Systems*, 27, 2321-2329.
[4] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529-533.
[5] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.
[6] Van Aken, D., Pavlo, A., Gordon, G. J., & Zhang, B. (2017). Automatic database management system tuning through large-scale machine learning. *Proceedings of the 2017 ACM International Conference on Management of Data*, 1009-1024.
[7] Zhang, J., Liu, Y., Zhou, K., Li, G., Xiao, Z., Cheng, B., ... & Liu, J. (2019). An end-to-end automatic cloud database tuning system using deep reinforcement learning. *Proceedings of the 2019 International Conference on Management of Data*, 415-432.
[8] Qin, X., Zhou, X., Chang, L., & Li, G. (2022). Autonomous vector database tuning: Challenges and opportunities. *IEEE Data Engineering Bulletin*, 45(2), 34-45.