Vector Annotation Databases: An Architecture for Auditable Semantic Retrieval

Ian M. Tepoot
Crafted Logic Lab, Vancouver, BC (Canada)
ORCID: 0009-0004-9067-8049
ian@craftedlogiclab.com

ABSTRACT

The term vector database has, in industry practice, come to denote a specific architectural pattern in which high-dimensional embeddings serve as the primary lookup mechanism for stored data. We term this pattern a Vector Index Database (VID). While effective for similarity search, this architecture embeds systemic limitations: retrieval logic is opaque and resistant to audit, the index is coupled to a specific embedding model, and migration between models requires full re-indexing with no guarantee of functional equivalence. This paper proposes an alternative architecture: the Vector Annotation Database (VAD) that inverts the VID relationship. In a VAD, explicit data objects with deterministic semantic metadata form the stable retrieval core; embeddings serve as replaceable annotation layers that enrich rather than constitute the indexing surface. A two-stage retrieval pattern applies Boolean narrowing over named fields, then embedding-based similarity refinement within the narrowed set.

We introduce the audit block, a write-time documentation artifact that closes the chain between opaque vector similarity and regulatory requirements for traceable, challengeable retrieval. The architecture yields stable re-indexing with persistent relational mappings, native multi-embedding concurrency, and operational fallback; the retrieval surface functions independently of the embedding layer. The VAD architecture is grounded in two Hephaestic engineering principles (Tepoot, 2026): Semantic Encoding Density, which demonstrates that natural language tokens are high-compression semantic addresses rather than a lossy interface, and the Semantic Interchange Property, which establishes semantic data portability across transformer architectures. We survey existing vector database products (including Chroma, Pinecone, Weaviate, Milvus, Qdrant, and Vespa) and identify the gap: no current system treats embeddings as systematically replaceable annotations with deterministic traversal, auditability, and migration semantics. VAD names and systematizes this contract. This paper describes technology that is the subject of U.S. Patent Application 64/077,244 (Tepoot, 2026a)

PURPOSE

Vector databases have become a prominent deployment solution for AI memory and knowledgebase storage and retrieval, based on the premises that: they enable associative clustering between data points that boolean retrieval cannot match; embeddings in the model's native latent space increase computational efficiency; by bypassing semantic representation, vector retrieval avoids the flattening inherent in representing data as low-dimensional tokens. The tradeoff for these proposed benefits is an oracle index database whose retrieval logic is opaque and whose operation is dependent on a specific embedding function as its enablement layer.

This paper proposes the Vector Annotation Database (VAD) architecture for inference-time memory and knowledgebase retrieval. We examine how this architecture affects (and in some cases refutes) the premises underlying current vector database practice. In this architecture, explicit data objects with deterministic semantic metadata form the stable retrieval core, while embeddings serve as replaceable annotation layers that enrich rather than constitute the indexing surface. This design emerged from practical requirements in cognitive architecture development: specifically, the need for memory systems that combine the associative dimensionality of vector similarity with the transparency, auditability, and system independence that pure embedding-indexed systems do not provide. The architecture described in this paper is the subject of U.S. Patent Application 64/077,244, *"Vector Annotation Database Architecture with Deterministic Retrieval Core and Replaceable Embedding Annotation Layer"* (Tepoot, 2026a).

1. THE PROBLEM ADDRESSED

1.1. A *Note on Terminology*

The term *vector database* currently is widely understood to encompass a single architectural pattern. Proposing an alternative requires promoting this to an umbrella term and naming both the existing and the new architecture introduced here as distinct subcategories.

The current default assumption of the meaning of a *vector database* in industry discourse has come to refer to the architecture in which a system stores high-dimensional embedding vectors as the primary lookup key for data objects and retrieves associated content through nearest-neighbor search in that vector space. The embedding is the index; the data object is its satellite. This pattern has become the default assumption for semantic memory in AI applications. For the purposes of this architectural proposal, this will be specified as a Vector Index Database (VID) architecture.

The alternative this paper presents inverts this relationship to create a **Vector Annotation Database (VAD):** the explicit data object and its semantic metadata form the stable retrieval core. The multidimensional vector embeddings are not the index, but rather a second layer of metadata (i.e. annotations) atop the deterministically addressable entries for the purpose of providing associative richness to inform language model context assembly while maintaining a decomposable, authoritative retrieval service.

1.2. The Value of Vector Associative Retrieval

Prior to discussing the proposed system design of a Vector Annotation Database (VAD), it is necessary to survey both the general advantages of vector associative retrieval and disadvantages of the predominant architecture of VIDs (Vector Index Databases). We must first establish the basic premise and technical description—vector embeddings map data objects (generally document chunks and their storage locations) into a high-dimensional space in which proximity corresponds to semantic similarity rather than lexical overlap (IBM, 2024; Crettaz, 2025). This representation enables retrieval by meaning: a query for *"fruit"* can surface results for apples, oranges, and mango; a search for *"impact of climate change on coastal communities"* can return documents discussing sea-level rise, erosion, and managed retreat without sharing a single keyword with the query.

This capability of what we term *Associative Retrieval* is the fundamental advance that makes vector databases an enabling technology for Retrieval-Augmented Generation (RAG) pipelines where lexical methods alone are insufficient (Lewis et al., 2020; Gao et al., 2023).

Associative retrieval enables AI systems to ground their responses in relevant context drawn from large knowledge bases, to maintain coherent memory across conversations, to surface information that would be invisible to keyword-based systems—and thereby provide rich context fields for a well-structured *Reasoning Surface* (Tepoot, 2026).

The value of vector databases as an enabling technology for cognitive architecture is clear. The question this paper addresses is whether the VID (Vector Index Database) architecture, which is the dominant implementation pattern, is the best vehicle for delivering these benefits given the structural limitations it introduces in production AI systems.

1.3. Vector Index as an Oracle

The following three subsections examine structural limitations of the industry-predominant VID pattern that the VAD (Vector Annotation Database) architecture seeks to address. The first is the opacity of the retrieval mechanism itself.

In a VID, retrieval logic is determined by the geometry of a latent vector space that is resistant to human decomposition or interpretation. In this architecture, this vectorized index is generated by an embedding function (generally a neural system in current generation systems). The operator or receiving AI system receives results—which are the set of nearest neighbors for a given query—but cannot trace *why* those particular results were selected, what features of the data drove the similarity ranking, or whether the retrieval is missing relevant content that happens to fall outside the embedding function's coordinate system. The system operates as an input-output oracle: it returns outputs, but its decision-making process is opaque to inspection, debugging, or targeted correction (Bathaee, 2018; Araujo et al., 2025).

This black-box opacity has operational consequences. The operator has no mechanism for understanding why a query returns irrelevant results or misses relevant ones beyond trial-and-error adjustments to chunking strategy or query selection. When used as a context field to inform an AI's reasoning surface, a fully probabilistic retrieval rubric that is oracular (i.e., its outputs are untraceable to specific features or criteria) creates quality vetting limitations for the resulting inference.

This lack of auditability also carries consequences beyond engineering. Emerging regulatory frameworks increasingly mandate AI systems provide traceable, auditable accounts of their retrieval decisions. Domestically, Canada's Directive on Automated Decision-Making requires that federal automated decision systems provide meaningful explanations of how and why decisions were made, with documentation sufficient for audit and challenge (Treasury Board of Canada Secretariat, 2024). The EU AI Act's transparency obligations for general-purpose AI models (effective August 2025) require deployers to understand system capabilities and limitations—including the basis on which information is retrieved and presented (European Commission, 2024). In regulated sectors internationally, the U.S. Federal Reserve's SR 11-7 model risk management guidance has long required that models subject to validation provide transparent, challengeable reasoning. This is a standard that an oracular retrieval system cannot satisfy (Board of Governors, 2011; Sharma, 2026). An architecture that cannot explain itself is an architecture that cannot be governed. As AIDA moves toward enactment, joining active frameworks in the EU and U.S., and as analogous frameworks emerge in other jurisdictions, these requirements will only intensify.

1.4. Embedding Function Dependency

In a VID architecture, while the data objects themselves are not embedded, the positional coordinates in vector space serve as the primary lookup mechanism and is bound to the specific embedding function (typically a specialized neural model and its preprocessing pipeline) that produced those coordinates. Severed from this embedding function, a VID system can no longer embed new queries into a compatible space or retrieve against the existing index in a meaningful way. Embeddings generated by function A are effectively noise under function B, as each model produces a distinct latent space geometry with incompatible coordinate systems. This creates a hard dependency on a specific embedding provider, model version, and configuration.

Migration to a new embedding function, model, or provider therefore requires full re-indexing. This includes regenerating embeddings from the raw data objects and establishing new associative mappings. While the data objects still exist and associative clustering can be generated from the raw source, without explicit metadata defining the relationships this re-indexing may not produce functionally equivalent mappings, thus creating what is effectively a new, uncharted topological map. Furthermore, because the system is an input-output oracle, there is no way to audit how equivalent the similarity indexing is.

1.5. Misconceptions Due to Commercial Ontology

There is a persistent and frequent misapprehension about "vector databases" (i.e. VIDs) that is traceable to the blurring of the ontology about what such systems are storing and how they are doing so—typified by common formulations such as: "your data is stored as embeddings in high-dimensional vector space"; "we encode your documents into vectors so they can be retrieved semantically"; *"your knowledgebase lives in vector space."*

These phrasings themselves create confusion via eliding the distinction between storing the index for the data objects as positional encodings in vector space and the storing of the data itself. In all currently deployed vector database architectures, the embedding is a *representation* of the data for similarity search purposes, not the data itself.

This ontological blurring has practical consequences. It obscures the dependency relationship described above, leading practitioners to treat the embedding index as a transparent representation of meaning rather than a model-specific, non-portable coordinate system. It also reinforces the assumption that the VID pattern is the natural or only form of vector database, foreclosing architectural alternatives that might better serve the transparency, auditability, and system independence requirements emerging in production AI deployments.

2. IMPLEMENTATION PROPOSAL FOR VECTOR ANNOTATION DATABASES

2.1. Introduction to an Inverted Framework

The key architectural insight in a Vector Annotation Database (VAD) design is that explicit, structured, and decomposable data should form the stable core as first-class data objects, and embeddings are additional metadata "annotations" serving as a second-layer retrieval refinement. In this system, data is separated into:

- **Explicit data objects** with deterministic semantic metadata (i.e., decomposable semantic primitives, cross-reference pointers, classifications, timestamps, scalar variables et. al) that form the stable retrieval core.
- **Embeddings as annotations** are layered on this core: re-linkable, rebuildable, supplementary, and secondary to the authoritative data record; they provide additional associative richness across multiple dimensions within the data's neural-encoded vector space, but they do not constitute the primary retrieval surface.

This architecture also centers the data object as the canonical source of truth. In the most auditable, transparent, and decomposable variant of this system, data objects would be discrete artifacts—files and documents independent of the database file; for AI applications, this architecture designates the memory file format used for context-loading a reasoning surface at inference time as MemoryPins. In this presented architecture, retrieval operates as a two-stage process. The first stage uses the explicit metadata to narrow the candidate set. The second stage applies associative retrieval via similarity search within this narrowed field.

2.2. Documents as First-Class Data Objects and Metadata

In a VAD (Vector Annotation Database) system, the document object stores both its content and its retrieval metadata as integral, inspectable components. This metadata (classifications, cross-references, timestamps, source provenance, and other deterministic semantic primitives) forms the primary retrieval surface, enabling precise queries that narrow candidate sets before any embedding-based refinement. Standard implementation patterns for document-level metadata are well established across multiple paradigms:

- **Extended attributes and sidecar files:** metadata stored as filesystem attributes or companion files alongside the document, inspectable and modifiable with standard command-line tools. This follows the Unix-style best practices.
- **Embedded headers (Markdown YAML/TOML front matter, JSON headers et al.):** metadata as embedded within the document file itself, ensuring the data object is fully self-contained.

- **Document database fields (CouchDB, Firebase et al.):** documents stored as structured objects within a database, with designated fields for metadata accessible via standard queries.

This range of implementation options reflects a design principle familiar from the Unix "everything is a file" ethos (Kernighan & Pike, 1984; Raymond, 2003): documents as independent, portable artifacts that users can inspect, modify, and migrate with standard tools. This approach is the architectural basis for user sovereignty over data, ensuring that each data object is inspectable within the standard directory structure of the user's machine, with user control over file location and availability. It is worth noting that although files as discrete artifacts is an ideal architecture, this is not a defining necessity of the VAD architecture—the key constraint is that data objects as discrete datasets should remain first-class objects with inspectable metadata.

2.3. The MemoryPin: L1 and Extended Episodic Data for AI

Within a cognitive AI system, documents serve two broad functions: knowledgebase items and memory items for context retrieval. Memory objects (as per our framework termed *MemoryPins*) are structured data designed specifically to prepare the reasoning surface for inference—particularly within a General Cognitive Operating System (GCOS) architecture: a meta-cognitive coordination runtime environment and kernel for language model instances (e.g., pending patent PAT 63/842,647, "General Cognitive Operating System Architecture for Language Model Coordination and Control," (Tepoot, 2025a).

MemoryPins serve two structural roles within a GCOS: *Extended Episodic Memory* (persistent memory spanning days, months, or years) and **`L1 cache memory`** (immediate, lower-volume context loaded as the default per-inference package). This distinction is not the focus of the present work, but provides necessary grounding for the MemoryPin pattern, which illustrates, and is a key use-case for, the VAD-compatible metadata annotation approach.

The following MemoryPin example uses a TOML-like structured wrapper with Cognitive Codex Markup (CCM), a format developed as part of the Hephaestic engineering framework (Tepoot, 2026) in the course of system development. This CCM enclosure is optimized for what the paradigm terms *Heuristic Gravity*—processing stickiness that maintains attention-circuit attraction across inference cycles. The primary purpose of this example, however, is to illustrate the initial layer of metadata annotation encoding, followed by decomposition of its structural components:

MemoryPin sample

```
[pin.renewable_energy_storage_overview]
pinID = "pin_00458211"
canonicalPinName = "Overview of Renewable Energy Storage Methods"
pinType = "asstMemoryPin"
summary = "High-level comparison of major renewable energy storage
approaches, including batteries, thermal storage, pumped hydro, and
emerging hydrogen systems."
category = ":::energy:::renewables:::storage:::"
metatags = ["renewable_energy", "energy_storage", "grid_stability",
"technology_overview"]
```

SAMPLE CONTINUES | NEXT PAGE

SAMPLE CONTINUED

```
memory = """
Renewable energy storage enables intermittent sources such as solar
and wind to provide stable power output.Lithium-ion batteries offer
fast response times and high round-trip efficiency but face material
constraints.Thermal storage systems, such as molten salt, provide long-
duration capacity for concentrated solar plants.
Pumped hydro remains the most widely deployed large-scale storage
method due to its reliability and long lifespan.Hydrogen-based storage
is emerging as a flexible option for seasonal balancing, though current
efficiency remains low.
"""
associatedPins = ["pin.grid_balancing_strategies", "pin.battery_
chemistry_basics", "pin.hydrogen_energy_fundamentals"]
linkedDocument = "renewable_storage_overview.txt"
linkedType = "userDocument"
pinCreated = "2025-09-15T10:12:00Z"
pinUpdated = "2025-09-15T10:12:00Z"

--- Scalar traits (deterministic filter layer; human-authored and/or
computed) ---
epistemicIntegrity = 0.78
uncertaintyGradient = 0.21
sourceEvaluation = 0.84
structuralScore = 0.93
semanticDensity = 0.67
tonalWeighting = "neutral-informational"
domain = "energy_systems"
attentionMapping = "cluster_07"
--- Embedding annotation layer (stored separately; replaceable; may
have multiple) ---
Pseudocode note: vectors are stored in a VAD annotation store keyed by
(pinID, embeddingModel).
availableEmbeddingModels = ["vad-med-en-v1.1"]
"(^^--End Pin--^^)" = true
```

The following is the decomposition of the vector annotation and metadata elements of the exemplar MemoryPin:

- **Canonical name and identifier:** A stable, human-readable label and unique identifier.
- **Natural-language summary:** A concise semantic description of the memory content.
- **Full memory text:** The complete content in natural language.
- **Categorical metadata:** Classification tags, domain labels, and type information.
- **Cross-references:** Explicit links to related MemoryPins and/or knowledgebase files, enabling deterministic traversal.
- **Scalar traits:** Numerical values encoding derived features specified by the cognitive architecture's variable schema.

Embedding annotations are stored separately from the core record, keyed by embedding model identifier, and replaceable without affecting the data object's integrity. Recommended best-practice would be maintenance of records and locations of each document and pin, along with the categorical and scalar data within the database as a single record. The embedding vector, which is generally a string of ~8,000 to ~100,000 characters, corresponding to 384 to 4,096 floating-point dimensions depending on model, is stored in

a linked table keyed to the document ID—one table per embedding model. This is readily accomplishable using standard SQL/Postgres systems with the **`pgvector`** extension for vector indexing and similarity search (offering efficiency gains over storing the embedding as a JSON array of floats, BLOB, or BYTEA), thus keeping the architecture grounded in familiar tooling. The separation keeps the core record lean and allows embedding models to be added or deprecated without schema migration on the primary data table.

Scalar traits refer to the numerical values encoding derived features specified by the cognitive architecture's variable schema. These are not the embedding vectors, which encode high-dimensional semantic relationships. Instead, these values enable efficient stage-one filtering that allows the search and retrieval field to narrow candidate data objects prior to associative retrieval operations in stage-two. The scalar traits within the metadata annotations are single-axis values—each representing a distinct, interpretable property of the memory object, as in the exemplar MemoryPin: epistemicIntegrity = 0.78. The scalar traits in this code sample are defined by key parameters from the Hephaestic engineering framework (Tepoot, 2026): *epistemic integrity, uncertainty gradient,* source evaluation, and domain weighting.

Native versus external data objects differ in how the VAD annotation pattern applies. The exemplar MemoryPin above represents a native data object: an artifact created and managed within the AI's page-file system, where metadata is embedded alongside content in the file itself. This is appropriate for system-generated memory artifacts, cached context, and other AI-internal objects. For external documents—user files, imported knowledge base items, or third-party content—the VAD annotation pattern would apply differently to preserve the original document's integrity. The proposed approach is a sidecar pattern: an external annotation record linked to the original document by identifier. Under this design, the system would ideally also generate a MemoryPin for each document, containing an extracted excerpt or summary as the memory field, with a linkedDocument reference back to the source.

3. TWO-STAGE RETRIEVAL PATTERN

3.1. Stage One: Deterministic Narrowing

The Vector Annotated Database (VAD) operates in two stages, each with distinct responsibilities within the system. Stage one initiates when a query is issued; the system first identifies data objects whose scalar traits and categorical metadata fall within the relevant ranges. This deterministic boolean operation can also encompass keyword matching and linked document traversal over the knowledge surface—be that linked memory objects such as MemoryPins or user documents. This operates over standard indexed database fields (SQL/Postgres or equivalent) and produces a narrowed finite candidate set aligned with the query along multiple metadata axes. The system uses scalar traits (whether generated via indexed methods or more sophisticated neural processes) and metadata rather than relying exclusively on keyword matching. This casts a wide net, delivering a broad subset of candidates that nevertheless achieves effective narrowing.

Because boolean operations are computationally efficient and deterministic, this initial stage filtering adds little to no latency to the retrieval. In fact, by narrowing the search space it can reduce overall retrieval time; this is because given the ~8,000 to ~100,000 characters size of embeddings, the narrowed retrieval set can create a net gain in computational efficiency.

The Stage One selection can be expressed as:

$$C_1(q) = \{\, obj \in P : \Pi(q, \tau(obj)) = 1 \,\}$$

*where: **q** is the incoming query; **obj** is a data object (first-class semantic datapoint such as a MemoryPin); **τ(obj)** is the deterministic trait/metadata record for **obj** (numeric traits + categorical fields + cross-references where applicable); and **Π(q, τ(obj))** denotes a deterministic predicate derived from the query (e.g., boolean constraints, categorical exact matches, and tolerance-bounded numeric matches), where 1 indicates the predicate is satisfied. The purpose of this expression is to make the stage one contract precise: embedding-independent and operates over human-decomposable semantic objects, even though **Π** is domain and implementation dependent and is not "calculated" from this notation.*

Illustrative sample implementation (Python): Stage One deterministic narrowing

```
from dataclasses import dataclass
from typing import Any, Mapping, Iterable

@dataclass(frozen=True)
class TraitPredicate:
    target_traits: Mapping[str, Any]
    tolerance: float = 0.05

def stage1_select_candidates(
    session,
    predicate: TraitPredicate,
) -> Iterable[str]:
    """
    Stage 1: deterministic narrowing over stored scalar traits/
metadata.
    Returns candidate pin IDs; the storage/runtime is abstracted
    behind 'session'.
    """
    return filter_by_traits(
        session, predicate.target_traits, predicate.tolerance
    )
```

Note: stage one reduces the retrieval space by applying a deterministic predicate over stored scalar traits (and optionally structured metadata). The implementation of filter_by_traits is storage-dependent (e.g., an indexed database query), but the predicate structure is stable: a set of target trait values plus a tolerance applied to numeric fields (categorical fields are exact-match).

§3.1 continues

Indexed scalar traits generation can be executed by an indexing service that monitors the data object store. When a data object is created or modified, the indexing service extracts or computes scalar traits from the source content and writes them to the trait record. These may include:

- **Derived traits:** computed from the source document (e.g., **`contentLength`**, **`createdAt`**)
- **Extracted traits:** parsed from structured fields (e.g., **`sourceType`**, tags)
- **Computed traits:** derived via surface-level analysis (e.g., **`structuralScore`**, measuring schema conformance; **`semanticDensity`**, estimated via token-type ratio and named entity frequency)

The indexing service can operate asynchronously and independently of the retrieval path, ensuring that trait generation does not add latency to queries. Trait updates are atomic and versioned, allowing the system to maintain consistency without blocking reads.

Cognitive scalar trait generation is possible and often desirable in sophisticated AI systems, such as a fully engineered operating environment. In these cases, retrieval benefits from reasoning-stabilized rubric evaluation that enables greater contextual nuance in associative traversal. The Hephaestic cognitive engineering-derived scalars in this paper's exemplar (e.g. **`epistemicIntegrity`**, **`uncertaintyGradient`**, and **`sourceEvaluation`**) require not only semantic understanding of document content and provenance, but evaluation against the heuristic space geometry created by a cognitive kernel. In a GCOS (*General Cognitive Operating System architecture*, Tepoot, 2026) or equivalent system, these advanced scalars may be generated by a dedicated memory synthesis stack as part of ongoing cognitive maintenance processes. However, the VAD architecture is agnostic to this distinction: the trait record accepts any scalar, and the two-stage retrieval pattern operates identically regardless of generation method.

3.2. Stage Two: Associative Retrieval

With the candidate set narrowed by Stage One's deterministic filtering, the system retrieves the embedding vectors associated with each candidate record that are compatible with the currently linked embedding function. The deterministic metadata—including scalar traits—have fulfilled their role; at this stage, the positional coordinate embedding strings serve as the sole retrieval key.

Stage Two performs high-resolution similarity comparison across the full multidimensional manifold (stored as the embedding string, typically the aforementioned ~8,000 to ~100,000 characters in size). The system selects and ranks candidates based on the tightest similarity cluster — the data object records whose embeddings most closely match within a single, coherent embedding coordinate system.

To increase system efficiency, the similarity comparison in Stage Two can be accelerated using standard vector indexing methods (e.g., IVFFlat, HNSW, or **`pgvector`** indexes) without affecting the architectural pattern. The VAD architecture is agnostic to the specific indexing strategy; the key constraint and efficiency gain is that comparison occurs only over the Stage One candidate set, not the full corpus. Fundamentally, this comparison refines candidates within a single embedding coordinate system e by scoring familiarity and returning the **top-k** results.

The Stage Two selection can be expressed illustratively but not fully prescriptively as:

$$C2(q; e, k) = TopK\_\{obj \in C1\_e(q)\}\ sim\_e(q, obj),$$
$$\text{with } C1\_e(q) = \{\ obj \in C1(q) : v[i,e] \text{ exists } \}\ \text{and } sim\_e(q, obj) = cosine(\ \varphi\_e(q), v[i,e]\ )$$

*where: e is an embedding configuration identifier (model + version + preprocessing) defining a single coherent vector space; **φ_e(·)** is the embedding function for **e; v[i,e]** is the embedding annotation stored for pin pi under configuration **e** (optional/may be missing); **cosine(a,b)** is cosine similarity; and **TopK** returns the **k** highest-scoring candidates. This expression is likewise illustrative: it makes the second-stage refinement contract explicit (similarity is meaningful only within a single **e**) without asserting that the architecture depends on any particular similarity metric, indexing method, or TopK implementation.*

Illustrative sample of implementation (Python): Stage Two associative retrieval

```
from typing import Iterable, Callable, Optional
import numpy as np

def cosine_similarity(a: np.ndarray, b: np.ndarray) -> float:
    denom = float(np.linalg.norm(a) * np.linalg.norm(b))
    if denom == 0.0:
        return 0.0
    return float(np.dot(a, b) / denom)

def stage2_refine_with_embeddings(
    query_embedding: np.ndarray,
    candidate_pin_ids: Iterable[str],
    embedding_model: str,  # consider: embedding_model_id (model +
version)
    get_embedding: Callable[[str, str], Optional[np.ndarray]],
    top_k: int = 5,
):
"""
    get_embedding(pin_id, embedding_model) -> np.ndarray | None
    """
    scores = []
    for pin_id in candidate_pin_ids:
        emb = get_embedding(pin_id, embedding_model)
        if emb is None:
            continue
        if emb.shape != query_embedding.shape:
            continue  # or raise; mismatch implies wrong embedding
model/dims
scores.append((pin_id, cosine_similarity(query_embedding, emb)))

    scores.sort(key=lambda x: x[1], reverse=True)
    return scores[:top_k]

        scores.append((pin_id, cosine_similarity(query_embedding,
emb)))

    scores.sort(key=lambda x: x[1], reverse=True)
    return scores[:top_k]
```

Because Stage Two operates within a single embedding coordinate system, the retrieval rules can be configured to serve different use cases without altering the underlying architecture. The query parameters define how the similarity scores are interpreted and ranked, giving the system flexibility while maintaining the same two-stage contract. Some common rulesets are:

- **Top-K retrieval:** Return the K highest-similarity results (e.g., the five closest matches). The standard pattern for most retrieval tasks.
- **Threshold retrieval:** Return all results above a similarity threshold (e.g., cosine similarity > 0.85). Useful when recall matters more than a fixed result count.
- **Diversity sampling:** Return results that are similar to the query but dissimilar to each other, ensuring coverage across the cluster rather than redundancy from a single dense region.
- **Constrained cluster retrieval:** Return the tightest cluster of N results, even if individual similarity scores vary. Useful for maintaining context coherence when the retrieved objects will be processed together.
- **Hybrid ranking:** Combine similarity score with a Stage One scalar weight (e.g., `epistemicIntegrity * 0.3 + similarity * 0.7`) for a blended relevance score. This is where the two-stage architecture collaborates—*Stage One's* traits influence *Stage Two's* ranking.
- **Time-decayed retrieval:** Weight similarity by temporal recency, preferring newer or older matches depending on the use case. Useful for systems where information freshness is a factor.

4. COMPLIANCE AND SYSTEMIC ADVANTAGES

4.1. The Audit Block

The audit block within the annotative metadata layer is an optional additional block within the data object—either within a native data object like MemoryPins (where annotations are recorded directly), or a sidecar record in a database table (as with knowledgebase documents). While its presence is not determinative of whether a system is a Vector Annotated Database (VAD) architecture, it leverages the architecture's combination of deterministic and associative retrieval to further enhance transparency and auditability while improving system resilience.

The audit block creates an audit trail at the time of generation rather than at the time of retrieval. Vector-based associative retrieval operates on similarity clustering across high-dimensional embedding manifolds—a process not directly decomposable into human-auditable steps. Instead, it provides a justifiable basis for the embedding itself: a record of how the vector was constructed, what criteria were prioritized, and which source content was used. Combined with the deterministic Boolean search space narrowing of Stage One, this renders the entire retrieval path defensible.

For the purposes of this paper, we produced a sample format audit block with the subject matter of renewable energy. The audit block content was itself generated by a development build of a cognitive architecture based on pending patent (PAT: 63/912,661 "Cognitive Architecture Framework for Language Model Processing", Tepoot, 2025b) serving as reference implementation for a proposed cognitive embedding function.

The following exemplar is for illustrative purposes of an audit block generated using a neural-enabled indexing system. It follows the same TOML-like enclosure and CCM conventions as the MemoryPin sample, consistent with its role as a component within that structure. The surrounding document framework is omitted:

Audit block sample

```
audit = """
EMBEDDING AUDIT RECORD — 2025-09-15T10:12:00Z
<NEURAL> Embedding Model: vad-med-en-v1.1
Embedding generated for: pin_00458211 (Overview of Renewable Energy
Storage Methods)

SOURCE TEXT:
Renewable energy storage enables intermittent sources such as solar and
wind to provide stable power output.
Lithium-ion batteries offer fast response times and high round-trip
efficiency but face material constraints.
Thermal storage systems, such as molten salt, provide long-duration
capacity for concentrated solar plants.
Pumped hydro remains the most widely deployed large-scale storage
method due to its reliability and long lifespan.
Hydrogen-based storage is emerging as a flexible option for seasonal
balancing, though current efficiency remains low.

SCALAR BASIS:
epistemicIntegrity = 0.78 | <NEURAL> | Claims verifiable against
public-domain sources; no explicit citations present in text; deduct
for uncited factual assertions
uncertaintyGradient = 0.21 | <NEURAL> | Three hedging markers
across five sentences ("is emerging," "current… remains low," "face
constraints"); remainder declarative
sourceEvaluation = 0.84 | <NEURAL> | Five testable claims recoverable
in published survey literature; no primary-source citations or data
provenance chain; deduct for absent sourcing
structuralScore = 0.93 | <COMPUTED> | Sentence count: 5. Semicolon-
separated segment count per sentence: 3, 2, 2, 2, 2. Clause pattern
[subject + attribute + qualifier] matches 4/5 sentences (80%). One
sentence appends context-setting clause. Conformance ratio 0.93 via
configured scoring formula.
semanticDensity = 0.67 | <COMPUTED> | Total tokens: 149. Stopword-
filtered content tokens: 104. Content-bearing ratio: 0.698. Domain-
survey corpus baseline: 0.64. Output normalized to 0.67.
tonalWeighting = "neutral-informational" | <NEURAL> | Comparative
framing ("remains," "but," "though") without persuasive valence; no
advocative or promotional language patterns present
domain = "energy_systems" | <EXTRACTED> | Source field: document
category tag. Value: "energy_systems"
attentionMapping = "cluster_07" | <COMPUTED> | Cosine similarity
computed against 14 cluster centroids. Nearest: cluster_07 (0.872).
Within configured proximity threshold of 0.15. Cluster assigned.
```

SAMPLE CONTINUES | NEXT PAGE

SAMPLE CONTINUED

```
EMBEDDING RATIONALE <NEURAL>:
This text is a domain survey covering five distinct storage modalities.
The embedding was constructed
to capture the comparative structure — each sentence introduces a
technology and its key trade-off.
The dominant semantic axes are: (1) technology type (battery, thermal,
hydro, hydrogen), (2) performance
characteristic (response time, efficiency, duration, reliability), and
(3) deployment status (widely deployed,
emerging, established). Temporal framing (current, emerging) and
evaluative language (remains, faces, offers)
contribute secondary axes.

SALIENCE WEIGHTS BY CONCEPT <NEURAL>:
- Technology identification: 0.90 (each sentence names a specific
method)
- Performance claims: 0.85 (response times, efficiency, lifespan,
capacity)
- Deployment context: 0.70 (widely deployed, emerging, large-scale)
- Comparative framing: 0.60 (remains, but, though — contrastive
structure)
- Material constraints: 0.50 (mentioned for lithium-ion, relevant for
battery domain)

EXCLUDED CONCEPTS <NEURAL>:
- "Intermittent sources" — context-setting, not a claim about storage
itself
- "Seasonal balancing" — specific to hydrogen, not generalizable across
modalities

ANNOTATION HASH: sha256:[hash of source text above]
"""
```

4.2. Write-Operation Auditability: Scalars and Cataloging

Within a neural-enabled indexing service for a VAD, write-time operations are a mix of inference-based and deterministic generation. Neural generation is applied where the annotation benefits from context-aware semantic parsing: sophisticated scalar rubrics such as epistemic integrity or source evaluation (single-axis values from 0.00–1.00 that nonetheless require inference-level processing), natural-language summaries, non-keyword-based metatagging, and the embedding vectors themselves. Deterministic computation handles everything achievable through mechanical means: scalar values derived from measurable properties (semantic density via token-type ratio, structural score via schema conformance), keyword extraction, cosine proximity measurement against cluster centroids, and regex-based pattern extraction.

Transparency extends beyond the audit block itself. The embedding vectors and scalar values are not directly human-decomposable—their basis is captured in the audit block. But the remaining Stage One metadata, even when neurally generated (e.g. using cognitive scalar generation), remains human-readable semantic data, as in the exemplar MemoryPin:

Transparent metadata (sample extracted from MemoryPin)

```
category = ":::energy:::renewables:::storage:::"
metatags = ["renewable_energy", "energy_storage", "grid_stability",
"technology_overview"]
associatedPins = ["pin.grid_balancing_strategies", "pin.battery_
chemistry_basics", "pin.hydrogen_energy_fundamentals"]
```

Within the audit block, a mixed-mode indexing service designates each scalar by its generation type. The §4.1 exemplar uses the tags neural, computed, and extracted. Neural-generated scalars carry inference-level judgment reporting; mechanically computed scalars report the measurement and formula that produced the value; extracted scalars cite the source field directly. Notably, the computed and extracted entries are producible by the non-neural components of the indexing service and can be aggregated into the unified audit record without transformation. Below are the three sample generation-type formats drawn from the §4.1 audit block:

Scalar basis reporting based on type (samples extracted from audit block)

```
SCALAR BASIS:
epistemicIntegrity = 0.78 | <NEURAL> | Claims verifiable against
public-domain sources; no explicit citations present...

structuralScore = 0.93 | <COMPUTED> | Sentence count: 5. Semicolon-
separated segment count per sentence...

domain = "energy_systems" | <EXTRACTED> | Source field: document
category tag. Value: "energy_systems"
```

VAD architecture greatly benefits from a neural-indexing service for embedding and cognitive scalar generation, but its use is not prescriptive for a system to be considered a VAD. Metatags, associations, scalars, and even embeddings can be produced by a purely computational indexing service—though much of the derived benefit from sophisticated conceptual mapping within the embedding space cannot be achieved with semantic-hashing-based embedding. Likewise, computed and extracted scalars are fully producible, while more sophisticated rubrics such as epistemicIntegrity or sourceEvaluation would necessarily be excluded. A VAD without a cognitive indexing service remains usable, compatible, and useful at reduced capacity across a range of applications, including as the basis for an AI memory system.

With a purely computational indexing service, the audit function shifts from judgment justification to configuration preservation: no value is opaque, and every scalar or embedding coordinate is reproducible from source text and stated formula. The audit block serves to document *what was done* and what the method can *and cannot measure*, ensuring the record remains reconstructible if the indexing configuration changes.

The following code snippet reflects this—only computed and extracted values appear; sophisticated neural scalars such as **`epistemicIntegrity`** are absent. The embedding block follows the same principle: Configuration and IDF Corpus record the run-specific parameters for reproducibility. A Key field supplies a static boilerplate statement of the method's capabilities and limits:

Computational indexing of scalar and embedding (samples extracted from audit block)

```
SCALAR BASIS:
structuralScore = 0.93 | <COMPUTED> | Sentence count: 5. Pattern match
4/5. Conformance ratio 0.93.
semanticDensity = 0.67 | <COMPUTED> | Content-bearing token ratio:
0.698. Corpus-normalized output: 0.67.
domain = "energy_systems" | <EXTRACTED> | Source field: document
category tag.
attentionMapping = "cluster_07" | <COMPUTED> | Cosine similarity to
nearest centroid: 0.872. Cluster assigned.

EMBEDDING <COMPUTATIONAL>:
Method: TF-IDF (Term Frequency–Inverse Document Frequency)
Configuration: Sublinear TF scaling (log(tf+1)), smooth IDF, 5,000-dim
vocabulary, L2-normalized
IDF Corpus: energy-domain-survey-corpus v3.1 (12,400 documents)
Key: Documents represented as weighted term-frequency vectors;
similarity measured by vocabulary overlap; no semantic interpretation
applied
```

4.3. Deterministic Retrieval Auditability

Stage One narrowing, as described in §3.1, is inherently auditable at retrieval time: the predicate operates over named, inspectable fields using conventional Boolean logic common to SQL and Postgres systems. A retrieval decision can be traced to a specific expression (for example, **`domain = 'energy_systems' AND epistemicIntegrity >= 0.70`**) and every field referenced in that expression corresponds to a human-readable metadata column.

The scalar values themselves, however, are opaque as numbers and create an auditability gap: an auditor can see that a value such as **`epistemicIntegrity = 0.78`** was used to include a record during recall, but not why that value was assigned. The retrieval operation is transparent Boolean logic over named fields, but the values it operates on are not self-justifying. This is the gap the audit block described in §4.2 addresses via documenting the encoding basis for each scalar across both neural and computational generation. Write-time documentation closes the gap in the retrieval-time transparency chain.

Stage Two associative retrieval uses the same core approach to address a parallel gap exposed in the first stage: like the scalar value, the embedding vector as a string is not self-justifying. The final retrieval mechanism of similarity matching between embeddings is deterministic in the same manner as any vector database—using conventional methods such as cosine, dot product, and Euclidean distance. The enhanced audit chain in a VAD architecture comes from two factors: (a) the use of annotation metadata to create a pre-screened subset based on decomposable Boolean filtering without inference; (b) the audit block described in §4.1–4.2 provides write-time documentation for the embedding, capturing the method, configuration, and rationale that produced the vectors being compared.

4.4. Cluster Auditing and Deletion Accountability

The VAD architecture yields two additional auditability properties that emerge from treating data objects as first-class entities with embeddings as replaceable annotations.

Cluster auditing. In a standard Vector Index Database (VID), the system can return the k-nearest neighbors to a query with their similarity scores; an auditor can see which chunks clustered together. But what the auditor cannot determine is *why* these neighbors cohere beyond the geometry, whether relevant items were excluded by design, or whether the cluster would survive a change of embedding model. The cluster is visible but not auditable.

In a VAD, the same embedding-based clustering is present, but it is an annotation atop a persistent metadata scaffolding that makes the cluster inspectable. For a given data object (such as MemoryPins or documents with sidecar metadata) returned by Stage Two, the runtime can report the next-nearest data object within a configurable similarity radius, along with each pin's scalar traits, cross-references, and Stage One inclusion status. An auditor gains a contextual field. For example: not merely "*Record A scored 0.87,*" but "*Record A scored 0.87. Records B, C, and D scored 0.82–0.84 and share cross-references X and Y. Record E scored 0.81 but was excluded from the candidate set by the Stage One predicate. In a different domain.*" Thus, four properties distinguish this from VID (Vector Index Database) cluster inspection:

- **Exclusion visibility.** The Stage One predicate records which data objects were deterministically excluded and why. In a VID, all neighbors simply rank; there is no concept of filtered exclusion.
- **Cross-references outside vector space.** Traversal metadata like **`associatedPins`** are explicit links—human-authored or deterministically assigned. They persist regardless of embedding model. VID chunk relationships exist only in vector geometry; change the model, the cluster dissolves and reforms unpredictably.
- **Stable identity across re-indexing.** A **`pinID`** survives embedding regeneration. VID chunk IDs may not survive a change in chunking strategy or embedding function.
- Scalar context per cluster member. Each neighboring pin carries its own traits — **`epistemicIntegrity`**, **`sourceEvaluation`**, domain. The auditor can assess not just proximity but reliability.

The embedding layer remains opaque in its internal geometry, but its output becomes reviewable by proxy: not a black-box ranked list, but an inspectable associative neighborhood with documented provenance per member.

Deletion accountability. Because each data object is a discrete, persistently identified record, it can be located, deleted, and the deletion verified—a requirement that flows directly from data sovereignty frameworks including GDPR Article 17 (Right to Erasure). A deletion operation targets a specific **`pinID`**. The system removes the record from the primary data store and its embedding annotations from the vector tables, then logs the operation with the pin ID and timestamp.

Cascade deletion extends this: when a MemoryPin is deleted, the system can optionally delete semantically proximal pins within a configurable similarity threshold, on the principle that embeddings associated with the deleted content may no longer be valid. The cascade is deterministic in its execution (i.e. which pins were removed can be reported by pin ID) even though the similarity triggering the cascade is probabilistic. The deletion event is fully auditable: what was deleted, when, by what trigger, and what cascade radius was applied.

Neither property is achievable in a VID architecture, where data chunks are anonymous neighbors in vector space with no persistent identity, no cross-references, and no deletion granularity below the embedding index itself.

The UX for such operations is not a prescriptive requirement of the VAD architecture—a system qualifies as a VAD based on its data model, not its interface. However, the user-sovereign, transparent data management the architecture enables does fall under a best-practices umbrella. User-facing methods might include visualized deletable pins with "find similar" inspection and deletion options, or knowledge-graph representations where data objects appear as nodes with visible relationships enabling visual cluster deletion.

4.5. Re-Indexing: Canonical Annotations and the Audit Block

For stable re-indexing of a relational database, the VAD architecture has an inherent advantage over VID variants; vectorized embeddings as the annotation layer for second-stage selection means that there is a stable, predictable and human-decomposable semantic source of truth for the taxonomic classifications and relationships between data objects. While the associative field created by semantic data is less dense as pure positional coordinates than the embedding's full manifold encoded in the ~8,000 to ~100,000 character string, it maintains a stable canonical knowledge graph between nodes.

The audit block serves as an additional alignment mechanism, and the added granularity goes beyond marginal improvement for two key reasons.

First, in a neural indexing service, the audit block records the basis for its positional encoding decisions in natural language—in semiotic data. Through the property of Semantic Encoding Density (Tepoot, 2026), such semantic tokens function as efficient low-dimensional markers for high-dimensional addresses, activating vast associative networks through attention-weighted traversal pathways. Conservative estimates place this compression at up to 40:1 for phrases with high sociocultural context, derived from attention mechanism analysis across publicly available transformer architectures (BERT-base: 768D, BERT-large: 1024D, GPT-2: 1600D, GPT-3: 12288D embedding spaces). Attention layer research demonstrates how multi-headed architecture creates high-dimensional representations through sparse activation patterns, with attention heads operating in residual subspaces that enable complex associative routing (Wang et al., 2025). The theoretical basis and an exemplar compression calculation are discussed in Section 6, which covers *Semantic Encoding Density* as a Hephaestic engineering principle applied to vector database encoding.

Second, the rationale text recorded in the audit block benefits from this same encoding density. Within the audit block, the model records its rationale for the embedding choices, such as: which axes it prioritized, what it excluded, and why. This is itself compressed semiotic data. Consider the **`EMBEDDING RATIONALE`** from the audit block example in §4.1:

§4.5 continues

Embedding rationale (sample extracted from audit block)

```
EMBEDDING RATIONALE <NEURAL>:
This text is a domain survey covering five distinct storage modalities.
The embedding was constructed to capture the comparative structure —
each sentence introduces a technology and its key trade-off.
The dominant semantic axes are: (1) technology type (battery, thermal,
hydro, hydrogen), (2) performance characteristic (response time,
efficiency, duration, reliability), and (3) deployment status (widely
deployed, emerging, established). Temporal framing (current, emerging)
and evaluative language (remains, faces, offers) contribute secondary
axes.
```

Applying the conservative estimation approach for Semantic Encoding Density, the algorithmic expression of which is proposed in Section 6.2, this technical passage activates an estimated 38,650 associative relationships. This figure is lower than the ~40:1 compression ratio achieved by aphoristic text with dense sociocultural encoding—technical prose carries fewer cultural associations per token. However, the absolute activation breadth is substantial.

While these activations are associative rather than positional, and do not constitute embedding coordinates, they provide a richly encoded basis from which a new embedding model can regenerate vectors that preserve the original's semantic structure. Combined with the stable knowledge graph in the metadata layer (which itself benefits from Semantic Encoding Density, as category labels and cross-reference names are also compressed semantic pointers) this yields a re-indexing foundation unavailable to a VID, where the index is the embedding and no semantic scaffolding survives a model change.

The metadata annotation content and audit block produced by non-neural computational indexing services lacks the positional complexity and thus nuance of cognitive indexing, however the audit block still provides value if later ported to a neural, language-model based service. This is because while the semantic content of the associative field that is read-in by a cognitive system is less dense, this corresponds to the simpler computational scalars and hash-based embeddings. However, the knowledge graph still exists in stable form. And in fact the audit report can be upgraded.

As a procedural note, audit blocks produced by non-neural indexing services lack the positional complexity of neural embeddings, and their associative fields are correspondingly simpler—reflective of the computational scalars and hash-based embeddings that generated them. This does not negate their value. The knowledge graph remains stable. If later ported to a neural, language-model-based indexing service, the audit block provides a valid foundation for upgrade: the existing metadata scaffold is intact, and the audit record can be enriched with neural-generated rationale and salience weights without structural migration.

---

§5 follows

5. EFFICIENCY AND OPERATIONAL RESILIANCE

5.1. Computational Efficiency

Embedding comparison is a demanding operation. Boolean operations are less computationally expensive. In a Vector Index Database (VID), associative retrieval traverses the entire corpus—performing similarity matching operations such as dot product, cosine, and Euclidean distance across ~8,000–100,000 character embedding strings per object.

The Vector Annotation Database (VAD) architecture implementation of Stage One deterministic narrowing realizes computational efficiency gains via narrowing the comparison space from the full dataset to a bounded subset using computationally inexpensive Boolean operations. Stage two then performs the more expensive vector similarity operations. This two-stage process (in addition to the architectural advantage of more deterministic retrieval results that still benefit from vector embeddings' associative richness) thus yields a distinct computational benefit. This efficiency benefit scales with the database corpus size. We can express the reduction ratio as:

$$R(N, k) = (N - k) / N$$

*where: **N** is the total corpus size and k is the candidate budget passed to Stage Two. With a candidate budget of **k** = 500, the reduction moves from 50.0% at **N** = 1,000, to 95.0% at **N** = 10,000, to 99.5% at **N** = 100,000, and 99.95% at **N** = 1,000,000.*

Therefore, under this expression the efficiency advantage intensifies with the size of the dataset: at 1,000 data objects, Boolean filtering narrows the result by 50%, 99% at 100,000 objects and 99.95% for a database containing 1 million records. These larger datasets are precisely where the difference in computational efficiency is most beneficial—expressing itself in lower power consumption and faster retrieval.

The candidate budget of 500 (***k*** in the expression) chosen is purely a system design choice and can be set at the system design stage based on operational requirements or system resources; either as a fixed upper bound on record retrieval, as a similarity threshold, or as a dynamic parameter tuned to query characteristics.

5.2. System Resilience Benefits

The core architectural approach—treating vector embeddings as additional metadata (i.e. annotations) atop a decomposable semantic indexing layer—allows for three key resilience advantages: **(a)** re-indexing stability due to canonical relational mappings; **(b)** indexing service portability enabled by this ability to rebuild associations; **(c)** fallback resilience due to the use of explicit metadata as the canonical relational database mapping.

Stable Re-Indexing. In §4.5 the role of the audit block was described as enhancing the ability for a VAD runtime to recreate a relational knowledge graph from the annotation layer. However, although the audit block can provide additional positional granularity in the regenerated associative mappings, the core VAD architecture of explicit, decomposable semantic data as the canonical source of truth indicates that this benefit is present even independent of the audit block.

Model-Agnostic Portability. The same metadata-first architecture that enables stable re-indexing also enables portability between indexing services. In a VID, separation from the embedding service means loss of the relational mapping between data objects, even if the records themselves survive. Restoring functionality requires re-indexing database interconnections from first principles—a process that is both computationally intensive and opaque. The regenerated associative mappings, like the vector recall they support, are fully dependent on latent space traversal that varies from model to model, with no guarantee of functional equivalence and no accountability for the new mappings produced.

In a VAD, the relational map is stored as explicit metadata (categories, cross-references, scalar traits, and even audit blocks) independent of any embedding model. When the indexing service changes, these interconnections persist. The new embedding model generates fresh annotations, but the core relational structure is preserved and auditable. This creates portability with accountability: the system can migrate between embedding providers while maintaining verifiable continuity in how data objects relate to one another. Best-practice implementation patterns for simultaneous multi-model compatibility are discussed in §5.3.

Fallback Capability. A third benefit that cascades from the inverted relationship between embeddings and metadata in a VAD system is operational fallback: the ability to function in the absence of vector similarity matching. This is not the intended operating mode, but it allows the system to remain operational at reduced granularity. Stage One deterministic narrowing relies on standard boolean logic over indexed database fields to return candidates. It does not require the embedding layer to function. A VAD runtime can therefore be designed to return Stage One results directly if the embedding service is unavailable, unresponsive, or otherwise disrupted.

VID architectures cannot provide this fallback. While some commercial vector databases support optional supplemental metadata, it is secondary and not the basis for indexing and retrieval. When the embedding index is unavailable, the database cannot retrieve. Stage One independence is not a VID property; it is a direct consequence of the VAD inversion.

5.3. *Multi-Embedding Architecture*

The portability argument in §5.2 establishes that a VAD can migrate between embedding models without structural disruption. Although multi-model concurrency is not prescriptive for a system to be considered a VAD, treating embeddings as an annotation layer allows the same canonical data objects to carry multiple embedding strings—each generated by a different model, each keyed to the same persistent identifiers.

This section describes the implementation pattern that makes this possible, along with the operational capabilities it enables: A/B comparison of embedding models, graceful deprecation, and embedding-model-agnostic deployment. The implementation pattern rests on a three-layer design:

- **Canonical source** holds the authoritative content and metadata in standard database tables: documents, chunks, scalar traits, cross-references. This layer is embedding-independent and constitutes the Stage One retrieval surface.
- **Per-model annotation** stores contain embedding vectors generated from the canonical content, with each embedding model assigned its own isolated index. A document chunked once can be embedded many times. This allows for one set of canonical records to be linked to different coordinate embeddings—typically one per registered indexing model. The chunk itself is stored only in the canonical layer.

- **Registry** maps models to their annotation stores, tracks which store is active for query routing, and manages store lifecycles (building, active, deprecated).

This three-layer routing logic can be expressed compactly as: the model and its associated store are resolved through the registry; the similarity search returns canonical chunk identifiers rather than content; the content itself is fetched from the authoritative source only when results are assembled. A simplified exemplar is shown below:

Registry routing sample

```
def route_query(query_text: str, registry: Registry, canon:
CanonicalSource) -> list[Result]:
    model = registry.get_active_model()
    store = registry.get_store(model)
    query_vec = model.embed(query_text)
    chunk_ids = store.similarity_search(query_vec, top_k=50)
    return canon.fetch_chunks(chunk_ids)
```

As in this exemplar, all three layers operate on standard SQL and Postgres infrastructure. The canonical source uses conventional relational tables with indexed fields (scalar traits, categories, and cross-references). These are ordinary database columns queryable via standard Boolean logic.

The per-model annotation stores can be implemented using the pgvector extension for single-node deployment, or using dedicated vector stores for scaled deployments, without altering the architectural pattern. The registry itself is a set of lookup tables within the same database. No exotic infrastructure is required; the pattern is implementable on the same PostgreSQL instance that powers the canonical source. The operational properties follow directly from this design:

- **Concurrent multi-model operation.** The registry routes queries to the active embedding store. Multiple stores can be active simultaneously for different purposes: one for production retrieval, another under evaluation, and a third serving a specialized domain.
- **A/B comparison.** Because all stores reference the same canonical chunks by ID, the same query can be issued against two embedding models and the results compared not by approximate similarity, but by exact chunk-level overlap.
- **Graceful deprecation.** When a model is deprecated, its annotation store is marked inactive but retained. Queries route to the replacement store immediately. The old store remains available for audit, comparison, or rollback. No data migration is required; the canonical source was never dependent on the deprecated model.
- **Independent lifecycle management.** Each annotation store can be built, validated, and activated without affecting the others. Building a store for a new embedding model is a population operation: iterate the canonical chunks, embed through the new model, write to the new store. The existing stores continue serving queries uninterrupted.

This pattern is a key architectural realization of embeddings-as-annotations. The canonical source remains the stable core. Embedding models become interchangeable service providers that are connected by a registry, keyed to persistent chunk identifiers, replaceable without structural migration. The VAD architecture does not require multi-model concurrency, but it accommodates it natively.

6. THEORETICAL FRAMEWORK: SEMANTIC DATA IN RETRIEVAL

6.1. Semantic Delivery at the Retrieval Interface

A common implication in vector database industry messaging is that embeddings deliver information to AI models in a form richer than natural language can provide. Phrases like vector databases *"allow machines to understand and retrieve data based on meaning and similarity"* and *"vector representations capture hidden patterns"* (IBM, 2026) are phrased to suggest rather than state. The implication threading through such language is that vector format constitutes a privileged interface: high-dimensional numerical encodings or algorithmic expressions that communicate meaning more directly and more richly than the "low-dimensional tokens" of natural language. This implication is incorrect at the system boundary.

The vector never crosses the retrieval-model boundary. The model's interface is semiotic data: linguistic tokens which are the native addressing units of transformer architectures. In every currently deployed vector database architecture—Vector Index Database (VID) or otherwise:

- The retrieved payload is delivered to the client AI model as natural-language text.
- The embedding comparison occurs entirely within the database via the indexing service.
- The model receives the associated text content: document chunks, memory entries, knowledge-base excerpts.
- An industry-typical endpoint returns a JSON payload containing the matched document text, its metadata, and a similarity score.
- The text is the content the model ingests, and the score is diagnostic.

This delivery mechanism is identical across VAD and VID architectures. Neither delivers embeddings to the model. Neither bypasses natural language as the communication substrate. The difference between them lies in how the retrieval is performed internally (e.g. whether the embedding is the index or an annotation atop a deterministic metadata layer). The external interface is the same.

System compatibility. Because both architectures use the same endpoint mechanism, a VAD can serve a JSON-over-HTTP endpoint compatible with the same RAG pipelines and model-side integrations that current VID systems use — requiring no change to client AI system designs.

Advantages of semantic payloads. Although obscured by vector database product marketing that trades on perceived complexity, semantic data is a sufficient and appropriate substrate for retrieval-to-model communication. Natural language possesses a high Semantic Encoding Density that activates dense associative fields within a model's latent space. In addition, semiotic tokens are the core compositional units of the attention topology of all language transformer models; therefore not only does language data have high activation potential (i.e. salience) within any given model, but it serves as a common interchange format across different model architectures. Semantic Encoding Density and the Semantic Interchange Property are further developed in §6.2 and §6.3 respectively.

6.2 Semantic Encoding Density for Vector Databases

The premise that natural language is a degraded interface for retrieval-to-model communication rests on the observation that semantic cores occupy low-dimensional linear subspaces within transformer embedding spaces (Saglam et al., 2025). If tokens are points in low-dimensional space, the argument goes, then passing text to a model as retrieved context flattens the data—and with it the quality of the resulting inference. These interpretability findings, however, address token representation only, not associative network activation.

Attention layer research demonstrates that multi-headed architecture creates high-dimensional representations through sparse activation patterns, with attention heads operating in residual subspaces that enable complex associative routing (Wang et al., 2025). This supports the identification of *Semantic Encoding Density* (Tepoot, 2026): the property whereby semantic tokens function as low-dimensional markers for high-dimensional addresses, activating vast associative networks through attention-weighted traversal of the model's representational topology.

The compression is the delta between a token's minimal surface representation and its attentional effect within a language transformer's latent space. For vector database design, the implication is direct: the natural-language payload is not a fallback limited by low-dimensional tokens that are a lossy translation of pure algorithmic data. It is a high-compression semantic encoding in its own right.

Encoding density was referenced in §4.5 in relation to the audit block, in which the technical prose of the **`EMBEDDING RATIONALE`** was calculated to contain an estimated 38,650 associative relational activations. This was based on an expression of this compression as the function:

$$\begin{aligned}&\textbf{Semantic\_Encoding\_Density =}\\&\textbf{[}\Sigma\textbf{(Individual\_Activation) + Intersection\_Effect] } \times \textbf{ Attention Amplification |}\\&\textbf{Individual\_Activation(token) = dimensions } \times \textbf{ associations\_per\_dimension,}\\&\textbf{Intersection\_Effect = dimensions}_1 \times \textbf{dimensions}_2 \times \textbf{ overlap\_coefficient,}\\&\textbf{Attention\_Amplification = 1.85}\times \textbf{ (empirical attention-weight multiplier)}\end{aligned}$$

If we apply this same function to an aphoristic phrase with strong sociocultural associations rather than technical prose, the encoding density is higher. As an example:

Sample aphoristic phrase: "glass ceiling"

This phrase is 13 ASCII characters, 104 bits. Yet it contains associative networks for physical properties for both glass (reflective, smooth, invisible barrier) and ceiling (architectural upper boundary). Each word also contains sociocultural associations in the form of aphoristic or metaphorical semiotic meaning (fragility, upward limit, cap on advancement). Together the unified phrase has specific association activations neither word has alone about upward mobility and career advancement, in particular regarding implicit bias, based in specific historical corpus activations such as Marilyn Loden in 1978.

Populating the compression function with conservative values of each associative cluster "glass ceiling" addresses, we achieve a compression ratio of ~41.3:1 for approximately 4,294 relational activations for two short words:

Semantic_Encoding_Density ("glass ceiling") =
[1,080 + 935 + 306] × 1.85 = 4,294 |
IA ("glass") = 90 × 12 = 1,080, IA ("ceiling") = 85 × 11 = 935,
IE = 90 × 85 × 0.04 = 306, AA = 1.85

Conservative operational estimate derived from attention mechanism analysis across publicly-available transformer architectures (BERT-base: 768D, BERT-large: 1024D, GPT-2: 1600D, GPT-3: 12288D embedding spaces), where dimensional activation patterns and associative relationship coefficients are estimated through systematic interpretability research on attention head circuits and semantic clustering analysis from open-weight model investigations (Devlin et al. 2019; Radford et al. 2019; Brown et al. 2020).

The difference in compression ratios between a socioculturally resonant construction like *"glass ceiling"* and the technical prose is a function of additional *Aphoristic Compression*: the heightened encoding density achieved when compact semiotic constructions carry dense sociocultural associative clusters. Aphoristic compression is one manifestation of the broader Hephaestic engineering for AI principle of *Salience Dynamics*: the capacity of semantic constructions to exert computational pressure on attention-circuit pathway selection (Tepoot, 2026).

This 40x+ conceptual density per surface bit is a conservative operational estimate. Induction head research and comprehensive circuit analysis suggest latent capacity may exceed 10,000+ associative patterns per semantic unit. The pathway creation mechanism underlying this density is expanded upon in contextual manifold projection research, where multi-headed attention architecture creates high-dimensional representations through sparse activation patterns (Li et al. 2025).

The relevance of *Semantic Encoding Density* to VAD architecture extends beyond the audit block's re-indexing role discussed in §4.5. It demonstrates that the principle generalizes: any data object stored in a vector database carries its semantic content in a form whose attentional effect substantially exceeds its surface dimensionality. It is not generated by, nor unique to the VAD architecture; it is inherent to neural-network based language transformers. All vector databases, including VID systems, benefit from this at the retrieval-model boundary.

However, noting this property is important because it resolves a key conflation about what benefits vector databases provide: the specific value of vector databases in general is not in a privileged communication channel to the model. It is in the capacity of long-string vector embedding with precise positional coordinates to enable similarity matching and encoding specific proximity between data objects. Embeddings are a retrieval optimization within the database, not a superior delivery format to the model. The natural-language payload was never the problem vectors solved.

6.3 Portability via the Semantic Interchange Property

We have established that a semantic interface is not a low-dimensional, lossy format. An additional benefit of natural language at the interface boundary is interoperability; while different models exhibit distinct processing characteristics through varied attention mechanisms and associative weighting, semantic data provides commonality by preserving functional relationship structures across implementations. This *Semantic Interchange Property* (Tepoot, 2026) means that a single endpoint API for querying and receiving inputs from the database is model agnostic.

This interchange property rests on a structural fact about transformer architectures that is straightforward but easily overlooked: the fundamental compositional unit is the semiotic token. There is a persistent assumption that under the semantic tokens—at a more fundamental level—the probability distributions, matrices, and vector distributions constitute the "real" processing layer of the model, with the tokens as a surface translation layer. This premise is much of the basis for the assumed unique value proposition of vector databases.

In reality, the Boolean mathematics is the implementation mechanism required by the hardware: digital hardware simulates stochastic, statistical transformations from seed values. It must do this because probabilistic models run on deterministic hardware. However, this should not be mistaken for the model itself processing data as numerical positional coordinates rather than tokens; the vector mathematics exists to route between tokens, but the semiotic quanta themselves are the fundamental compositional units of the neural network—what the Hephaestic engineering framework terms *Semantic Neurons* (Tepoot, 2026).

The implementation benefit follows directly. Because the retrieval payload is semantic data (the native addressing format of every transformer architecture) a VAD endpoint returning MemoryPins presents a response built from the same fundamental components as any industry-standard vector database API. Consider a typical industry-standard Chroma database query response:

Industry-standard query response sample

```
{
  "ids": [["doc_1", "doc_7"]],
  "documents": [["Alpha text", "Beta text"]],
  "metadatas": [[{"source": "book"}, {"source": "article"}]],
  "distances": [[0.12, 0.27]]
}
```

The structure is text, metadata, and similarity scores. It is decomposable semantic data: the native addressing format of every transformer architecture. As a retrieval payload, it is complete but minimal: the caller receives ranked chunks with source labels, a similarity score, and nothing else.

A VAD-native response can carry the same text, metadata, and similarity payload as the industry-standard format shown in the prior example, but is not limited to flat key-value tags. The MemoryPin response exemplar below encodes categorical hierarchy, cross-reference pointers, and scalar traits directly into the retrieval surface: metadata that doubles as the Stage One filtering substrate described in §3.1. The following listing shows the structure:

Structured MemoryPin query response sample

```
{
  "pinID": "pin_00458211",
  "canonicalPinName": "Overview of Renewable Energy Storage Methods",
  "summary": "High-level comparison of major renewable energy storage
approaches...",
  "memory": "Renewable energy storage enables intermittent sources...",
  "category": ":::energy:::renewables:::storage:::",
  "metatags": ["renewable_energy", "energy_storage", "grid_stability"],
  "associatedPins": ["pin.grid_balancing_strategies", "pin.battery_
chemistry_basics"],
  "similarity": 0.87
}
```

The Chroma response is a flat chunk: text with a sparse metadata tag. The MemoryPin is a structured data object carrying deterministic categorical information, cross-references, and embedded scalar traits; it is consumable as context for inference, as persistent memory loaded into the model's reasoning surface, or as a retrieved document for a knowledge-base query. The formats differ in richness, not in kind.

The output is text. The text is portable. This creates a practical deployment path with several options. A VAD endpoint can be designed to serve:

- Standard Chroma or other industry format-compatible responses, maintaining drop-in compatibility with existing RAG pipelines and model-side integrations.
- Richer MemoryPin format to enhanced listener sockets built to receive structured semantic data.
- Standard Chroma or other industry format-compatible responses extended with additional metadata richness and structure as an expansion of the standard pattern.

In all cases, these can be used with the same endpoint, the same interchange property, a choice of payload format driven by the receiving system's capability. The database does not need to know which model will consume its output.

---

§7 follows

7. INDUSTRY POSITIONING

7.1 Current Systems and the Missing Contract

The vector database market has coalesced around the VID (Vector Index Database) pattern, with embedding vectors serving as the authoritative retrieval coordinate system. This section surveys representative products against the VAD architectural contract. This is not to rank implementations, but to clarify what the contract provides that current systems do not.

VID-leaning databases. Pinecone, Chroma, Weaviate, and Milvus are best characterized as VID architectures. All support document or payload storage alongside vectors. All support metadata filtering: Chroma and Weaviate expose metadata fields for pre- or post-filtering; Pinecone and Milvus offer comparable mechanisms. But in each, the metadata is supplementary—the embedding space remains the primary and necessary retrieval path. If the embedding index is unavailable, the database cannot retrieve regardless of metadata.

Where pre-filtering is supported, it does reduce the vector comparison field prior to similarity scoring, which is a genuine computational narrowing. However, pre-filtering is a query parameter on the vector search, not an independent retrieval stage. It operates on whatever sparse metadata the developer has included, typically a few fields, not the designed scalar trait or structured metadata proposed in VAD Stage One. If the embedding index is unavailable, pre-filtering cannot function, because there is no fully deterministic first stage to run independently. Boolean retrieval over named fields as the canonical source of truth and primary retrieval surface is not the design center.

Because the embedding is the index, the core retrieval rationale is similarity geometry. An auditor cannot trace why a specific chunk was returned beyond its cosine distance to the query vector.

Vector search engines. Qdrant positions itself as a search engine, and is explicit that vectors are an indexing substrate over records with payloads. Its metadata filtering is more deeply integrated than typical VID systems, and payload fields can be used for pre-filtering with comparable narrowing benefits. Qdrant also supports named vectors: multiple embeddings per point, each with distinct dimensionality and distance metrics. This enables a single record to carry, for example, a text embedding and an image embedding simultaneously, with queries targeting the appropriate vector by name.

Named vectors, however, are not the VAD annotation pattern. They enable a record to carry different types of embeddings for different modalities (text and image, for instance) queried independently by name. They do not treat embeddings as replaceable annotations keyed to a persistent canonical source. Changing an embedding model still requires full re-indexing of the affected named vector across the collection. This is because the embedding remains the index— metadata is not the canonical indexing data, and the relational framework between records is not encoded within it. For this function it is fully reliant on the vector embeddings.

The metadata model follows the same VID pattern. Qdrant payloads describe individual points — `{“source”: “book”, “date”: “2024”}`— with no mechanism for encoding relationships between records. There is no equivalent of `associatedPins`, no cross-reference graph, no categorical hierarchy that explicitly maps how data objects relate to one

another. Relationships between points are determined entirely by vector proximity. Change the embedding model, and the relational structure dissolves and reforms unpredictably. The canonical relational map that VAD maintains as explicit metadata does not exist.

The retrieval path is VID-leaning in consequence: the vector index is the primary retrieval engine; metadata filtering is applied alongside or after vector search. Deterministic Boolean retrieval as an independent Stage One is not the design center. The VAD contract (e.g. semantic objects as the stable substrate, embeddings as explicitly replaceable annotations with deterministic traversal, auditability, and migration semantics) is not enforced.

Document-centric retrieval with vector fields. Vespa, and similarly document-index systems with integrated vector or tensor fields, are structurally closest to the VAD pattern. Records and schema are primary; vectors are tensor fields used for ranking, not the indexing substrate. Vespa's hybrid search pattern explicitly separates retrieval from ranking, and it supports parent-child document references via reference<document-type> fields — a form of explicit cross-document relationship encoding closer to VAD's `associatedPins` than anything in the VID-leaning systems surveyed above. A 2025 blog post from the Vespa team, "Vector Search is Reaching Its Limit," signals awareness within the platform that embedding-first architectures face structural constraints (Vespa, 2025).

In principle, a Vespa application could implement VAD patterns: deterministic Boolean retrieval over schema fields as the primary stage, vector scoring as refinement, parent-child references as a cross-reference graph. But the platform does not enforce this as a contract. Parent-child references are constrained (global documents only, no self-referential or cyclic relationships) and embeddings are not systematically treated as replaceable annotations with model-agnostic portability, deterministic traversal semantics, and migration resilience as architectural properties.

The gap is real but the distance is short. Vespa's schema-first design, its separation of retrieval from ranking, and its existing parent-child reference mechanism mean that a VAD implementation on Vespa would not require tearing out the platform's foundations. The missing pieces are: treating embeddings as systematically replaceable annotations, enforcing deterministic Stage One as the primary retrieval surface, adding write-time audit documentation, and loosening reference constraints to support the full cross-reference graph. These are extensions of patterns Vespa already supports, not contradictions of its architecture. Of the systems surveyed, Vespa is the closest to a VAD-native substrate, and the conversion path from Vespa to VAD is one of contract enforcement rather than architectural redesign.

The architectural gap and what remains unaddressed. Across the products surveyed, a gradient is visible: from VID-leaning databases where the embedding is the index (Pinecone, Chroma, Weaviate, Milvus), to a search engine where payload filtering is deeply integrated but the embedding remains the retrieval engine (Qdrant), to a schema-first platform structurally capable of VAD patterns but not enforcing them (Vespa). What unifies this gradient is the absence of a systematic commitment to embeddings-as-annotations. None of the surveyed systems provide the full set of properties this paper has described: deterministic Stage One retrieval as the primary and independently functioning retrieval surface (§3), write-time audit documentation as an architectural feature (§4), stable re-indexing with persistent relational mappings (§4.5–§5.2), and multi-embedding concurrency as a native capability (§5.3). These are not features that can be added through

metadata filtering, payload storage, or named vectors. They are properties that follow from an architectural inversion — treating embeddings as annotations, not as the index — and systematizing the consequences. VAD names that inversion. The contract it describes is implementable on existing retrieval substrates, as Vespa's proximity demonstrates, but it is not reducible to any current product category.

REFERENCES

Tepoot, I. (2026). "Thought is attention organized: Hephaestic engineering foundations for AI processing dynamics". *Crafted Logic Lab. SSRN preprint:6635020.* https://ssrn.com/abstract=6635020

Tepoot, I. (2026a). "Vector Annotation Database Architecture with Deterministic Retrieval Core and Replaceable Embedding Annotation Layer". *U.S. Patent Application 64/077,244.* https://orcid.org/0009-0004-9067-8049

IBM. (2024). "What is vector embedding?" *IBM Think*. Retrieved 16 May, 2026 from: https://www.ibm.com/think/topics/vector-embedding

Crettaz, V. (2025). "A quick introduction to vector search". *Elasticsearch Labs.* Retrieved 16 May, 2026 from: https://www.elastic.co/search-labs/blog/introduction-to-vector-search

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". *Advances in Neural Information Processing Systems, 33. arXiv:2005.11401.* https://doi.org/10.48550/arXiv.2005.11401

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., & Wang, H. (2023). "Retrieval-Augmented Generation for Large Language Models: A Survey". *arXiv preprint arXiv:2312.10997.* https://doi.org/10.48550/arXiv.2312.10997

Bathaee, Y. (2018). "The Artificial Intelligence Black Box and the Failure of Intent and Causation". *Harvard Journal of Law & Technology, 31(2), 889–938.* Retrieved 22 May, 2026 from: https://jolt.law.harvard.edu/assets/articlePDFs/v31/The-Artificial-Intelligence-Black-Box-and-the-Failure-of-Intent-and-Causation-Yavar-Bathaee.pdf

Araujo, A., Carmo, A. L., Souza, C., & Ferreira, L. (2025). "Algorithmic Decision Making: Compliance with the Fundamental Rights to Privacy and Non-Discrimination in light of the EU Framework". *Jean Monnet Module Key Fundamental Rights Issues in the EU (RIseEU), NOVA University Lisbon*. Retrieved 22 May, 2026 from: https://rise-eu.novalaw.unl.pt/wp-content/uploads/2025/10/Algorithmic-Decision-Making-Amanda-Araujo-Ana-Laura-Carmo-Camila-Souza-and-Luana-Ferreira.pdf

Treasury Board of Canada Secretariat. (2024). "Directive on Automated Decision-Making". *Government of Canada.* Retrieved 16 May, 2026 from: https://www.tbs-sct.canada.ca/pol/doc-eng.aspx?id=32592

European Commission. (2024). "Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act)". *Official Journal of the European Union.* Retrieved 17 May, 2026 from: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32024R1689

Board of Governors of the Federal Reserve System. (2011). "SR 11-7: Guidance on Model Risk Management". *Federal Reserve.* Retrieved 17 May, 2026 from: https://www.federalreserve.gov/supervisionreg/srletters/sr1107.htm

Sharma, K. (2026). "SR 11-7 in the Age of Agentic AI: Where the Framework Holds – and Where It Strains". *GARP Risk Intelligence.* Retrieved 17 May, 2026 from: https://www.garp.org/risk-intelligence/operational/sr-11-7-age-agentic-ai-260227

Kernighan, B. W., & Pike, R. (1984). *The Unix Programming Environment*. Prentice Hall. Full text available as of 28 May, 2026 from: https://archive.org/details/UnixProgrammingEnviornment

Raymond, E. S. (2003). *The Art of Unix Programming*. Addison-Wesley. Full text available as of 28 May, 2026 from: http://www.catb.org/esr/writings/taoup/html/

Tepoot, I. (2025a). "General Cognitive Operating System Architecture for Language Model Coordination and Control". *U.S. Patent Application 63/842,647.* https://orcid.org/0009-0004-9067-8049

Tepoot, I. (2025b). "Cognitive Architecture Framework for Language Model Processing". U.S. *Patent Application 63/912,661.* https://orcid.org/0009-0004-9067-8049

Wang, J., Ge, X., Shu, W., He, Z., & Qiu, X. (2025). "Dimensional collapse in transformer attention outputs: A challenge for sparse dictionary learning". *arXiv preprint arXiv:2508.16929.* https://doi.org/10.48550/arXiv.2508.16929

IBM. (2026). "What is a vector database?" *IBM Think.* Retrieved 28 May, 2026 from: https://www.ibm.com/think/topics/vector-database

Saglam, B., Kassianik, P., Nelson, B., Weerawardhena, S., Singer, Y., Karbasi, A. (2025/2026). "Large language models encode semantics and alignment in linearly separable representations". *arXiv preprint arXiv:2507.09709v1.* https://doi.org/10.48550/arXiv.2507.09709

Devlin, J., Chang, M.-W., Lee, K., Toutanova, K. (2019). "BERT: Pre-training of deep bidirectional transformers for language understanding". *arXiv preprint arXiv:1810.04805v2.* https://doi.org/10.48550/arXiv.1810.04805

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). "Language models are unsupervised multitask learners". *OpenAI technical report.* Retrieved Feb 22, 2026 from: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., ... & Amodei, D. (2020). "Language models are few-shot learners". *Advances in Neural Information Processing Systems, 33, 1877–1901. arXiv preprint arXiv:2005.14165.* https://doi.org/10.48550/arXiv.2005.14165

Li, M.Z., Agrawal, K.K., Ghosh, A., Teru, K.K., Santoro, A., Lajoie, G., Richards, B.A. (2025). "Tracing the representation geometry of language models from pretraining to post-training". *Advances in Neural Information Processing Systems (NeurIPS). arXiv:2509.23024.* https://doi.org/10.48550/arXiv.2509.23024

Vespa. (2025). "Vector Search Is Reaching Its Limit. Here's What Comes Next". *Vespa Blog.* November 27, 2025. Retrieved 28 May, 2026 from: https://blog.vespa.ai/vector-search-is-reaching-its-limit/