Auxiliary Index Structures for Contextualized.pdf

by Turnitin Student

Submission date: 31-May-2025 02:16PM (UTC-0400)
Submission ID: 2689164965
File name: Auxiliary_Index_Structures_for_Contextualized.pdf (305.39K)
Word count: 8585
Character count: 55392

Auxiliary Index Structures for Contextualized Retrieval in RAG Systems: Design, Theory, and Applications

Mateus Yonathan
Software Developer & Independent Researcher

https://www.linkedin.com/in/siyoyo/

***Abstract*—This paper identifies a gap in the application of established data management concepts to the emerging field of Retrieval-Augmented Generation (RAG) systems. While temporal databases, data provenance systems, and information retrieval have been extensively studied in their respective domains, their integrated application to RAG presents unique challenges and opportunities. Vector databases excel at similarity search but often lack the governance capabilities needed for enterprise deployment. We explore how auxiliary index structures might complement vector databases by synthesizing concepts from these well-established fields. Drawing on temporal database principles, we consider time-aware indexing for historical retrieval. Inspired by data provenance systems, we examine relationship tracking between generated outputs and source materials. Borrowing from information security, we explore context isolation approaches. This theoretical exploration presents a mathematical formulation that connects these adapted concepts while acknowledging the significant overlap with existing technologies. The paper's contribution lies not in introducing novel algorithms but in identifying how these established concepts can be specifically tailored to address the unique operational challenges of enterprise RAG systems.**

I. Introduction

Modern Retrieval-Augmented Generation (RAG) systems leverage vector databases like Qdrant, Faiss, and Milvus for efficient similarity search capabilities [1], [2]. These vector databases provide robust functionality for semantic search and information retrieval applications [3]–[5].

While vector databases excel at finding similar content, they face practical challenges in real-world enterprise applications. Imagine trying to answer questions like: "Which version of a document was used to generate this AI response last month?" or "How do we trace which parts of our knowledge base influenced a particular output?" This paper explores how concepts from temporal databases, data provenance systems, and information retrieval might be adapted to address these challenges in RAG contexts.

We examine supplementary index structures (which we call "auxiliary structures") that could potentially work alongside vector databases to address four common concerns:

- **Temporal Information Management:** Drawing on concepts from temporal databases [6], [7], we consider how time-aware indexing might enable "point-in-time" retrieval in RAG systems. This could help answer questions based on what was known at a specific point in time [8], [9].
- **Lineage Tracking:** Adapting approaches from data provenance systems [10], [11], we explore how clear connections might be established between AI outputs and source materials. Such approaches could support compliance, accuracy verification, and debugging [12], [13].
- **Change Impact Analysis:** Inspired by version control systems like Git [14], we examine how incremental update tracking might help identify which embeddings need updating when source documents change. This could potentially reduce unnecessary reprocessing of entire datasets [15], [16].
- **Context Isolation:** Building on information security concepts, we consider how separate knowledge spaces might be maintained for different departments, clients, or use cases, extending approaches from context-aware information retrieval [17], [18].

By synthesizing these established concepts, we explore a potential approach that could be represented as:

$$\mathcal{RAG}_{\text{aux}} = (V, M, T, L, \Phi) \tag{1}$$

Where:

- V - The vectors stored in vector databases (the embedded content)
- M - Metadata about each vector (like source, owner, permissions)
- T - Time information for each vector (when it was created or modified)
- L - Links between vectors and source documents
- Φ - Signatures that might help with verification and matching

This representation adapts established concepts from different domains to the specific context of RAG systems. It's worth noting that many vector databases already provide metadata capabilities that could potentially implement aspects of this approach.

This paper explores:

- How these concepts from different domains might be adapted to RAG systems
- Potential operations they could enable (like time-based retrieval and change tracking)
- How they relate to existing information management approaches
- Implementation considerations for practical deployment

We take a pragmatic approach focused on real business problems, drawing connections to established techniques like metadata indexing, version control, and information retrieval. Our goal is to show how these existing approaches might be combined and adapted to address emerging challenges in RAG systems.

The remainder of this paper is organized as follows. Section VIII reviews related work in vector indexing, semantic retrieval, and data lineage. Section II presents a theoretical system model. Section IV details a mathematical formulation adapting concepts from different domains. Section V explores potential applications. Section VI discusses implementation considerations. Section VII provides a critical assessment, and Section IX concludes with reflections on this theoretical exploration.

II. System Model

This section presents a theoretical conceptualization of components and relationships that could potentially complement existing RAG systems. This model explores how auxiliary structures might relate to the core functionality of vector databases.

A. RAG Pipeline Components

A typical RAG system might include several interconnected components:

- **Document Corpus** D: The set of source documents providing knowledge to the system. These documents typically change over time.
- **Embedding Model** $E : D \to V$: A function mapping documents to vectors in a high-dimensional space. Documents are typically chunked before embedding.
- **Vector Store** $S = (V, I)$: A database storing vectors V with an index I for similarity search.
- **Retrieval Function** $R : q \to \{v_1, v_2, \ldots, v_k\} \subset V$: Given a query q, returns the k most similar vectors.
- **Generation Model** $G : (q, \{v_1, v_2, \ldots, v_k\}) \to r$: Given a query q and retrieved vectors, generates a response r.

B. Operational Considerations

In certain deployment scenarios, several operational considerations might arise:

- **Temporal Consistency**: Some use cases might benefit from accessing information consistent with the knowledge available at a specific point in time.
- **Traceability**: In some contexts, tracing system responses to specific document versions could be helpful for analysis and debugging.
- **Update Efficiency**: Selective updating of affected embeddings when documents change could potentially optimize resource usage in certain scenarios.
- **Context Management**: Some multi-tenant deployments might benefit from additional mechanisms to manage distinct knowledge contexts.

C. Theoretical Extended Model

One way to conceptualize these operational considerations is through an extended system model:

$$\mathcal{RAG}_{\text{extended}} = (D, E, S, R, G, A) \tag{2}$$

Where A represents potential auxiliary structures. These theoretical structures could maintain relationships and metadata that might complement vector store capabilities:

$$A = (V, M, T, L, \Phi) \tag{3}$$

In this theoretical framework:

- V represents the set of vectors stored in the vector database.
- $M : V \to P(K \times Val)$ could map vectors to sets of key-value metadata pairs, potentially capturing information such as source URL, user ID, document type, etc.
- $T : V \to \mathbb{N}$ could provide temporal indexing by mapping vectors to version or timestamp information.
- $L : V \to D$ could establish connections between vectors and their source documents.
- $\Phi : (v, m) \in V \times M \to \mathcal{S}$ could project vectors and their metadata to a semantic signature space $\mathcal{S}$.

D. Possible Operations

Such auxiliary structures might enable operations like:

- **Historical Retrieval**: $R_t(q) = \{v \in R(q) \mid T(v) \leq t\}$, potentially retrieving vectors relevant to query q that existed at or before time t.
- **Source Identification**: $L(R(q)) = \{d \in D \mid \exists v \in R(q) : L(v) = d\}$, possibly identifying source documents that contributed to a response.
- **Targeted Updates**: When a document changes, identifying vectors linked to that document could help prioritize regeneration.
- **Context Filtering**: $R_c(q) = \{v \in R(q) \mid \exists (k, val) \in M(v) : k = \text{“context”} \wedge val = c\}$, potentially filtering vectors by context.

E. Integration Possibilities

If implemented, such auxiliary structures might integrate with existing RAG components through various interfaces:

- **Vector Store Integration**: Auxiliary structures could potentially maintain references to vectors while leveraging the vector store's core functionality.
- **Document Processing**: When documents are processed, auxiliary structures could capture relationships between documents, chunks, and embeddings.

- **Query Processing**: Auxiliary structures might enhance retrieval by providing additional context or temporal consistency.
- **Analysis Tools**: Such structures could potentially provide interfaces for examining system responses.

This theoretical model offers one possible framework for thinking about auxiliary structures. Different implementations might prioritize different aspects depending on specific use cases and requirements.

III. Design Considerations

Implementing auxiliary index structures for production RAG systems requires careful consideration of architectural principles and design decisions. In this section, we explore key design considerations that influence the effectiveness, efficiency, and maintainability of these structures.

A. Separation of Concerns

The first principle is to maintain a clear separation between the vector database's primary responsibility (efficient similarity search) and the auxiliary structures' responsibilities (governance, traceability, and contextual enrichment):

- **Non-Invasive Integration**: Auxiliary structures should augment rather than replace existing vector database functionality. This allows organizations to leverage their existing investments in vector database technology while adding the necessary operational capabilities.
- **Clear Functional Boundaries**: Each component of the auxiliary structure (M, T, L, Φ) should have a well-defined responsibility, avoiding functional overlap that could lead to inconsistencies or maintenance challenges.
- **Layered Architecture**: Implement auxiliary structures as a layer above the vector database, providing a cohesive API that combines vector similarity search with the enhanced capabilities of contextual retrieval.

B. Performance Considerations

Auxiliary structures must be designed with performance in mind to avoid becoming bottlenecks in the RAG pipeline:

- **Tiered Indexing Strategy**: Implement a multi-tiered approach where frequently accessed metadata and temporal information are stored in high-performance in-memory structures, while less frequently accessed lineage information may reside in more cost-effective storage.
- **Lazy Loading**: Defer loading and computing expensive components (such as detailed lineage information or semantic signatures) until they are specifically requested, rather than loading everything for every query.
- **Caching Strategy**: Implement intelligent caching for commonly accessed metadata and temporal snapshots, with cache invalidation tied to document update events.
- **Query Optimization**: Design queries to filter based on metadata early in the retrieval process, reducing the number of vectors that need to be processed for similarity comparison.

C. Consistency and Durability

Ensuring consistency between the vector database and auxiliary structures is critical for system reliability:

- **Transaction Boundaries**: Define clear transaction boundaries that encompass both vector insertion/update and the corresponding auxiliary structure updates, ensuring atomicity of operations.
- **Eventual vs. Strong Consistency**: Consider whether eventual consistency is acceptable for certain auxiliary structures (e.g., analytics-focused metadata) or whether strong consistency is required (e.g., for access control metadata).
- **Failure Recovery**: Design for recovery from failures, ensuring that auxiliary structures can be rebuilt from vector database content if necessary, or vice versa.
- **Versioned Operations**: Implement operations as versioned changes rather than in-place updates, enabling point-in-time recovery and historical analysis.

D. Scalability Patterns

Production RAG systems must scale to handle large document corpora and high query volumes:

- **Horizontal Partitioning**: Design auxiliary structures to support horizontal partitioning based on logical boundaries (e.g., tenant ID, document category), enabling independent scaling of different segments.
- **Incremental Computation**: Implement semantic signatures and other derived structures as incremental computations that can be updated efficiently when documents change, rather than requiring full recomputation.
- **Asynchronous Processing**: Offload non-critical updates (such as analytics-related metadata) to asynchronous processing pipelines, avoiding impact on the critical retrieval path.
- **Hierarchical Storage**: Implement a hierarchical storage approach where recent and frequently accessed data is kept in high-performance storage, while historical data gradually moves to more cost-effective storage tiers.

E. Security and Privacy by Design

Auxiliary structures often contain sensitive metadata and lineage information, requiring robust security measures:

- **Encrypted Metadata**: Implement encryption for sensitive metadata fields, with access controlled through appropriate key management.
- **Access Control Granularity**: Design access control mechanisms with appropriate granularity, allowing fine-grained permissions on specific metadata fields or document categories.
- **Audit Logging**: Incorporate comprehensive audit logging for all access to and modifications of auxiliary structures, especially those related to governance and compliance.
- **Data Minimization**: Apply data minimization principles to metadata and lineage information, storing only what is necessary for the intended purposes.

F. Extensibility and Adaptability

RAG systems and their requirements evolve over time, necessitating flexible and extensible auxiliary structures:

- **Schema Evolution**: Design metadata schemas to accommodate evolution over time, using approaches like schema versioning or flexible document schemas.
- **Pluggable Components**: Implement auxiliary structures with clearly defined interfaces, allowing components to be replaced or extended without disrupting the entire system.
- **Feature Flags**: Use feature flags to gradually roll out new auxiliary structure capabilities, allowing controlled testing in production environments.
- **Backward Compatibility**: Maintain backward compatibility for APIs and data formats, ensuring that existing applications continue to function as auxiliary structures evolve.

G. Observability and Diagnostics

Effective monitoring and diagnostics are essential for maintaining auxiliary structures in production:

- **Performance Metrics**: Capture and expose metrics on auxiliary structure operations, such as metadata lookup time, lineage tracing latency, and semantic signature computation time.
- **Health Indicators**: Define clear health indicators for auxiliary structures, such as consistency between vector database and lineage mappings, or currency of temporal indices.
- **Diagnostic Interfaces**: Provide diagnostic interfaces that allow operators to verify the state of auxiliary structures and their relationships with the vector database.
- **Anomaly Detection**: Implement automated anomaly detection for auxiliary structures, identifying potential issues such as missing lineage information or inconsistent metadata.

H. Integration with Existing Infrastructure

Auxiliary structures must integrate effectively with an organization's existing data infrastructure:

- **ETL Pipeline Integration**: Design for integration with existing Extract, Transform, Load (ETL) pipelines, ensuring that document updates trigger appropriate updates to auxiliary structures.
- **Identity System Integration**: Integrate with existing identity and access management systems for consistent enforcement of access controls across auxiliary structures.
- **Observability Stack Integration**: Ensure that metrics, logs, and traces from auxiliary structures integrate with the organization's existing observability stack.
- **Backup and Disaster Recovery**: Incorporate auxiliary structures into existing backup and disaster recovery processes, with appropriate consideration for consistency between vector database backups and auxiliary structure backups.

I. Design Patterns for Specific Use Cases

Different deployment scenarios may require specialized design patterns:

- **Multi-Region Deployment**: For globally distributed systems, consider designs that replicate auxiliary structures across regions with appropriate consistency guarantees.
- **High-Compliance Environments**: In highly regulated environments, implement additional layers of verification and validation for auxiliary structures, possibly including cryptographic proof of lineage.
- **Edge Deployment**: For edge computing scenarios with limited resources, design lightweight versions of auxiliary structures that focus on essential capabilities while minimizing resource usage.
- **Hybrid Search**: When implementing hybrid search (combining vector and keyword search), design auxiliary structures that effectively bridge these different paradigms.

J. Summary

The design considerations presented in this section provide a framework for implementing auxiliary index structures that are robust, performant, and maintainable in production environments. By addressing concerns such as separation of responsibilities, performance optimization, consistency guarantees, scalability, security, extensibility, observability, and integration, organizations can create auxiliary structures that effectively support the operational requirements of enterprise RAG systems.

These design considerations will inform our mathematical model and implementation constraints in subsequent sections, ensuring that our theoretical framework translates into practical and effective solutions for real-world deployment scenarios.

IV. Mathematical Model

This section adapts established mathematical concepts from temporal databases, data provenance systems, and information retrieval to the context of RAG systems. Rather than introducing new mathematical structures, we explore how existing formalisms might be applied and combined in this specific domain.

A. Vector Representation and Metadata

Let $V \subset \mathbb{R}^d$ be the set of vector embeddings stored in the vector database, where d is the dimensionality of the embedding space. Each vector $v \in V$ represents the semantic content of a document chunk.

Drawing from metadata management systems, we can represent the metadata function $M : V \to P(K \times Val)$ that maps each vector to a set of key-value pairs, where K is the set of possible keys and Val is the set of possible values. This common approach to metadata might enable contextual information to be associated with each vector:

$$M(v) = \{(k_1, val_1), (k_2, val_2), \ldots, (k_n, val_n)\} \quad (4)$$

In practice, metadata might include information such as:

- Document identifiers: (key: "doc_id", value: "doc123")
- Source information: (key: "source_url", value: "example.com/doc")
- User or tenant context: (key: "tenant_id", value: "tenant456")
- Access information: (key: "access_level", value: "restricted")
- Document type: (key: "doc_type", value: "technical_manual")

Such metadata could potentially enable filtering and contextualization of search results beyond vector similarity. It's worth noting that many vector databases already support metadata storage and filtering capabilities that implement this concept.

B. Temporal Indexing

Adapting concepts from temporal databases [6], we can represent a temporal indexing function $T : V \to \mathbb{N}$ that assigns a version or timestamp to each vector, indicating when it was created or last modified:

$$T(v) = t \tag{5}$$

where t represents a temporal identifier (e.g., a Unix timestamp or a logical version number).

Such a function, common in temporal database systems, might enable operations like:

- **Point-in-time Retrieval**: For a query q and time t, $R_t(q) = \{v \in R(q) \mid T(v) \leq t\}$ could retrieve vectors that existed at or before time t.
- **Temporal Filtering**: For times $t_1 < t_2$, $\{v \in V \mid t_1 \leq T(v) \leq t_2\}$ might identify vectors created or modified within a specific time range.
- **Change Tracking**: For a document d at times $t_1 < t_2$, $\{v \in V \mid L(v) = d \wedge t_1 \leq T(v) \leq t_2\}$ could identify embeddings derived from document d that changed between times t_1 and t_2.

In some applications, temporal indexing might support:

- Generating responses based on historical knowledge states
- Analyzing changes to the knowledge base over time
- Tracking document representation evolution
- Supporting time-based querying for certain use cases

C. Lineage Mapping

A lineage mapping function $L : V \to D$ could connect each vector to its source document:

$$L(v) = d \tag{6}$$

where $d \in D$ would be the document from which vector v was derived.

This concept could potentially be extended to capture more granular relationships:

$$L_{ext} : V \to D \times C \times P \tag{7}$$

where C might represent document chunks and P might represent processing steps applied to generate the embedding.

Such mapping could potentially enable:

- **Source Tracing**: For a retrieval result $\{v_1, v_2, \ldots, v_k\}$, the corresponding source documents might be identified as $\{L(v_1), L(v_2), \ldots, L(v_k)\}$.
- **Impact Analysis**: For a modified document d, affected vectors could be identified as $\{v \in V \mid L(v) = d\}$.
- **Provenance Tracking**: For a generated response $r = G(q, \{v_1, v_2, \ldots, v_k\})$, the provenance chain might be represented as $\{L(v_1), L(v_2), \ldots, L(v_k)\}$.

In some systems, similar mapping could support:

- Investigating incorrect responses by examining source documents
- Updating embeddings when source documents change
- Providing attribution in generated responses
- Supporting access controls

D. Semantic Signature Projection

A semantic signature projection $\Phi : (v, m) \in V \times M \to \mathcal{S}$ could map vectors and their metadata to a semantic signature space $\mathcal{S}$:

$$\Phi(v, M(v)) = s \tag{8}$$

where $s \in \mathcal{S}$ would be a representation capturing semantic properties.

Such a signature might serve multiple purposes:

- **Semantic Change Detection**: For a document d that changes from version d_1 to d_2, comparing $\Phi(v_1, M(v_1))$ and $\Phi(v_2, M(v_2))$ (where v_1 and v_2 are the corresponding embeddings) might detect semantic shifts.
- **Anomaly Identification**: Outliers in the signature space could potentially indicate unusual content.
- **Filtering**: A compact signature space might enable prefiltering before more computationally expensive similarity calculations.

In certain applications, such signatures might help with:

- Detecting when content updates substantially change meaning
- Identifying potential quality issues
- Optimizing retrieval performance
- Supporting semantic categorization

E. Theoretical Operations

The model suggests several potential operations:

1) Insert Operation: When a new document d is added to the corpus:

$$\begin{aligned} \text{Insert}(d) = \{(v, m, t, d, s) \mid\ & v = E(d), \\ & m = \text{ExtractMetadata}(d), \\ & t = \text{CurrentTime}(), \\ & s = \Phi(v, m)\} \end{aligned} \tag{9}$$

This operation would create vectors, metadata, timestamps, lineage mappings, and semantic signatures for the document.

2) Update Operation: When a document d changes to d':

$$\begin{aligned}\text{Update}(d, d') = \{(v', m', t', d', s') \mid \\ v' = E(d'), \\ m' = \text{UpdateMetadata}(M(v), d'), \\ t' = \text{CurrentTime}(), \\ s' = \Phi(v', m'), \\ \forall v \in V : L(v) = d\}\end{aligned} \tag{10}$$

This operation would identify vectors derived from document d and update them based on d'.

3) Delete Operation: When a document d is removed from the corpus:

$$\text{Delete}(d) = \{(v, \text{LogicallyDeleted}) \mid v \in V : L(v) = d\} \tag{11}$$

This operation might mark vectors derived from document d as logically deleted (potentially preserving them for historical queries).

4) Historical Retrieval Operation: For a query q and time t:

$$\begin{aligned}\text{HistoricalRetrieve}(q, t) = \{v \in R(q) \mid \\ T(v) \leq t \wedge v \text{ not LogicallyDeleted}\}\end{aligned} \tag{12}$$

This operation might retrieve vectors relevant to query q that existed at or before time t and have not been logically deleted.

F. Considerations for Real-World Scenarios

This theoretical model suggests approaches to several scenarios that might arise in information retrieval systems:

- **Concurrent Updates**: When multiple updates occur simultaneously, temporal indexing could help ensure proper sequencing.
- **Embedding Changes**: As embedding models evolve, semantic signatures might help detect shifts in representation.
- **Retrieval Accuracy**: The combination of vector similarity, metadata filtering, and semantic signatures could provide multiple verification stages.
- **Document Versioning**: Lineage mapping combined with temporal indexing might help track document evolution.
- **Multi-tenant Scenarios**: Metadata-based filtering could support tenant boundary enforcement.

This mathematical model presents one theoretical approach to conceptualizing auxiliary structures for RAG systems. Different implementations might adapt these concepts based on specific requirements, existing infrastructure, and performance considerations.

V. Practical Applications

This section explores theoretical applications where auxiliary index structures might be relevant. We present these as conceptual scenarios rather than concrete implementation requirements, acknowledging that many of these applications could be addressed through existing approaches.

A. Historical Querying

Vector databases typically support filtering by timestamp metadata. The temporal indexing concept explored in this paper presents an alternative perspective on managing historical states:

- **Conceptual Approach**: A temporal index could potentially map versions of embeddings across time, allowing for retrieval operations that reference specific knowledge states.
- **Theoretical Use Case**: A question like "What did we know about product X as of January 2023?" might be approached through temporally-aware retrieval.
- **Potential Implementation**: Such functionality could be implemented in various ways, including through existing vector database metadata filtering or dedicated temporal indices depending on specific requirements.

B. Provenance Tracking

Understanding the relationship between generated outputs and source documents represents a common information management concern:

- **Conceptual Approach**: A lineage mapping could potentially connect embeddings back to their source documents, allowing for retrieval of provenance information.
- **Theoretical Use Case**: When investigating a generated response, it might be useful to identify which specific document versions contributed to that response.
- **Potential Implementation**: This capability could be implemented through various approaches, including metadata tracking within vector databases or separate mapping structures.

C. Content Evolution Analysis

Tracking how document representations change over time represents an interesting analytical perspective:

- **Conceptual Approach**: By maintaining temporal and lineage information, it might be possible to analyze how document representations evolve.
- **Theoretical Use Case**: Analyzing how a product description has changed semantically over multiple versions could provide insights into messaging evolution.
- **Potential Implementation**: Such analysis could be performed using various approaches, including comparing document versions directly or analyzing their embedding representations.

D. Multi-context Management

Information systems often need to manage multiple overlapping but distinct knowledge contexts:

- **Conceptual Approach**: Metadata-based filtering could potentially help manage boundaries between different knowledge contexts.
- **Theoretical Use Case**: A system serving multiple departments might need to maintain separate knowledge contexts while allowing controlled sharing of information.
- **Potential Implementation**: This capability could be implemented through metadata filtering in vector databases or through separate context-specific indices.

E. Update Optimization

When source documents change, determining which derived artifacts need updating represents a common workflow challenge:

- **Conceptual Approach**: Lineage mapping between documents and embeddings could potentially help identify which embeddings need regeneration when documents change.
- **Theoretical Use Case**: When updating a set of documents, it might be more efficient to regenerate only the embeddings directly affected by those changes.
- **Potential Implementation**: This optimization could be approached through various mechanisms, including document-embedding mappings or content-based change detection.

F. Semantic Shift Detection

Detecting when document updates significantly change meaning could be valuable in certain contexts:

- **Conceptual Approach**: Semantic signatures could potentially help detect when document updates result in significant meaning changes.
- **Theoretical Use Case**: Flagging when a product description update substantially changes the semantic content might be useful for review processes.
- **Potential Implementation**: Such detection could be implemented through embedding comparison, specialized semantic signatures, or other approaches to measuring semantic difference.

G. Cross-referencing Applications

Understanding relationships between different document versions might be useful in certain analytical contexts:

- **Conceptual Approach**: Combining temporal indices and lineage mappings could potentially enable cross-referencing between document versions.
- **Theoretical Use Case**: Identifying all documents that referenced a specific concept at a particular point in time might be useful for certain analytical tasks.
- **Potential Implementation**: This capability could be implemented through various approaches, including metadata filtering, specialized indices, or query-time processing.

H. Summary

The conceptual applications discussed in this section represent theoretical scenarios where auxiliary index structures might be relevant. These applications could be implemented in various ways depending on specific requirements, existing infrastructure, and performance considerations. Many of these capabilities might be achievable through existing vector database features, particularly through effective use of metadata filtering and application-level logic.

VI. Implementation Considerations

If auxiliary structures were to be implemented, various design considerations might influence their practical realization. This section explores theoretical implementation considerations, using C#/.NET concepts as illustrative examples of potential approaches.

A. Type Safety and Data Integrity

Type safety mechanisms could help maintain data integrity in auxiliary structures:

- **Type-Safe Identifiers**: Implementation might benefit from strongly typed identifiers to prevent accidental mixing of different identifier types:

```
public record DocumentId(Guid Value) { }
public record VectorId(Guid Value) { }
public Dictionary<VectorId, DocumentId>
    LineageMap { }
```

  This approach could help prevent confusion between different ID types.
- **Immutability Considerations**: For historical records, immutability might help prevent inadvertent modifications:

```
public record VectorRecord(
  VectorId Id,
  ImmutableArray<float> Embedding,
  ImmutableDictionary<string, string>
      Metadata,
  DateTimeOffset Timestamp) { }
```

  Record types with value-based equality could be suitable for representing versioned data in some implementations.

B. Version Management

Temporal queries and historical analysis might benefit from version-aware collection types:

- **Timestamped Snapshots**:

```
public record TemporalIndex<TKey, TValue>(
  DateTimeOffset Timestamp,
  ImmutableDictionary<TKey, TValue> Entries
      ) { }
```

- **Append-Only Structures**: For tracking lineage, append-only data structures might help preserve history:

```
public class AppendOnlyLog<T> {
  private readonly List<T> _entries = new()
      ;
  public IReadOnlyList<T> Entries =>
      _entries.AsReadOnly();
  public void Append(T entry) => _entries.
      Add(entry);
}
```

C. Data Representation

The choice of data representation formats could affect both usability and performance:

- **Human-Readable Formats**: For debugging purposes, readable formats might be beneficial:

```
[JsonSerializable(typeof(VectorRecord))]
public partial class AppJsonContext :
    JsonSerializationContext { }
```

- **Compact Storage Formats**: For efficient storage, more compact formats might be considered:

```
public class VectorRecordCompact {
  public string Id { get; set; } = string.
      Empty;
  public float[] Embedding { get; set; } =
      Array.Empty<float>();
  public Dictionary<string, string>
      Metadata { get; set; } = new();
  public long Timestamp { get; set; }
}
```

- **Complex Key Management**: For mappings with complex keys, custom serialization approaches might be needed:

```
public class KeySerializer<TKey> {
  public string SerializeKey(TKey key) =>
      JsonSerializer.Serialize(key);
  public TKey DeserializeKey(string
      serialized) =>
    JsonSerializer.Deserialize<TKey>(
        serialized)!;
}
```

D. Concurrency and Caching

In multi-user environments, concurrency management would likely be important:

- **Concurrent Access**:

```
public class ConcurrentIndex<TKey, TValue>
    {
  private readonly ReaderWriterLockSlim
      _lock = new();
  private Dictionary<TKey, TValue> _data =
      new();

  public void Add(TKey key, TValue value) {
    _lock.EnterWriteLock();
    try {
      _data[key] = value;
    }
    finally {
      _lock.ExitWriteLock();
    }
  }
}
```

- **Caching**: For frequently accessed data, caching mechanisms might improve performance:

```
public async Task<TValue> GetWithCaching<
    TKey, TValue>(
  TKey key, Func<TKey, Task<TValue>>
      valueFactory, TimeSpan expiration) {
  string cacheKey = key.ToString();
  // Implementation would depend on
      specific caching infrastructure
  return await valueFactory(key);
}
```

E. Consistency Management

Maintaining consistency across related data structures would be a key consideration:

- **Transaction Patterns**:

```
public class UnitOfWork : IDisposable {
  private readonly List<Action> _actions =
      new();
  private readonly List<Action>
      _compensations = new();
  private bool _committed = false;

  public void RegisterAction(Action action,
      Action compensation) {
    _actions.Add(action);
    _compensations.Add(compensation);
  }

  public void Commit() {
    foreach (var action in _actions)
      action();
    _committed = true;
  }

  public void Dispose() {
    if (!_committed) {
      foreach (var compensation in
          _compensations)
        compensation();
    }
  }
}
```

- **Optimistic Concurrency**:

```
public class VersionedEntity<T> {
  public required T Data { get; set; }
  public required string Version { get; set
      ; }

  public bool TryUpdate(T newData, string
      expectedVersion, out string
      newVersion) {
    if (Version != expectedVersion) {
      newVersion = Version;
      return false;
    }
    Data = newData;
    newVersion = Guid.NewGuid().ToString();
```

```
        Version = newVersion;
        return true;
      }
    }
```

F. Vector Database Integration

Integration with existing vector databases would likely require careful API design:

- **Abstraction Layers**:

```
public interface IVectorStore {
  Task<string> StoreVector(
    float[] embedding, Dictionary<string,
        string> metadata);
  Task<IReadOnlyList<SearchResult>> Search(
    float[] queryVector, int limit,
        Dictionary<string, object> filter);
}
```

- **Extension Methods**:

```
public static class VectorStoreExtensions {
  public static Task<IReadOnlyList<
      SearchResult>> SearchWithTimestamp(
    this IVectorStore store,
    float[] queryVector,
    int limit,
    DateTimeOffset asOfTime) {
    var filter = new Dictionary<string,
        object> {
      ["timestamp"] = new Dictionary<string
          , object> {
        ["$lte"] = asOfTime.
            ToUnixTimeMilliseconds()
      }
    };
    return store.Search(queryVector, limit,
        filter);
  }
}
```

G. Performance Optimization

Balancing feature richness with performance would require optimization strategies:

- **Deferred Loading**: Lazy loading patterns could help avoid unnecessary computation:

```
public class LazyLoader<TKey, TValue> {
  private readonly Lazy<Dictionary<TKey,
      TValue>> _data;

  public LazyLoader(Func<Dictionary<TKey,
      TValue>> factory) {
    _data = new Lazy<Dictionary<TKey,
        TValue>>(factory);
  }

  public TValue GetValue(TKey key) {
    return _data.Value[key];
  }
}
```

- **Incremental Processing**: Incremental update strategies might improve efficiency:

```
public class DeltaTracker<TKey, TValue> {
  private Dictionary<TKey, TValue>
      _baseline = new();
  private Dictionary<TKey, TValue> _changes
      = new();

  public void TrackChange(TKey key, TValue
      value) {
    _changes[key] = value;
  }

  public void CommitChanges() {
    foreach (var kvp in _changes)
      _baseline[kvp.Key] = kvp.Value;
    _changes.Clear();
  }
}
```

H. Summary

The implementation of auxiliary index structures would depend heavily on specific requirements, existing infrastructure, and performance considerations. The examples presented in this section are conceptual illustrations rather than specific recommendations. Different deployment scenarios would likely require different approaches to balancing features, performance, consistency, and integration with existing systems.

If implemented, auxiliary structures would need to be tailored to the specific needs of each deployment context, considering factors such as scale, query patterns, update frequency, and existing data management infrastructure.

VII. Critical Analysis

This section critically examines our exploration, acknowledging that many of the concepts we've discussed represent adaptations of existing approaches rather than novel innovations. We recognize substantial overlap with existing technologies and that many practical implementations would likely extend current tools rather than implementing our theoretical model from scratch.

A. Significant Overlap with Existing Systems

1) Vector Database Metadata Capabilities: Modern vector databases already provide robust metadata storage and filtering capabilities that implement many aspects of what we've described. For example, Qdrant supports payload data attached to vectors, and Milvus enables scalar filtering based on metadata fields [19]. These existing capabilities might address many of the use cases we've discussed without requiring separate auxiliary structures:

- **Temporal Management via Metadata**: Vector databases already allow storing timestamps as metadata and filtering based on them. Many applications could implement the temporal indexing function T we described using these existing metadata capabilities without additional structures.
- **Lineage via Document References**: Standard metadata fields can store document identifiers, which already enable basic lineage tracking in many implementations.

The lineage mapping L we presented can often be implemented directly through these existing reference mechanisms.

- **Context Isolation via Filters**: Many vector databases support namespace concepts or metadata filtering that effectively creates isolated contexts. These features might already address the context isolation requirements we've discussed.

2) Direct Equivalents in Version Control Systems: Git and similar version control systems already provide sophisticated mechanisms for tracking changes that parallel many aspects of our temporal modeling:

- **Commit History**: Git's commit history directly implements temporal versioning, potentially making our temporal indexing function redundant for document versioning.
- **Blame and Log**: Git's blame and log functionality track lineage and provenance in ways that might be more comprehensive than our proposed lineage mapping.
- **Branching**: Git's branching model already provides mechanisms for managing parallel contexts that might be more sophisticated than our context isolation approach.

3) Information Retrieval Indices: Traditional IR systems use inverted indices to map terms to documents. These established approaches share conceptual similarities with the structures described in this paper:

- **Indexing Approach**: Inverted indices map terms to documents, while the structures described here relate vectors to their metadata, timestamps, and source documents. Both approaches aim to enhance retrieval capabilities.
- **Temporal Aspects**: Classic inverted indices generally don't incorporate temporal dimensions as a core feature, though they can be extended to do so. The temporal indexing discussed in this paper explicitly incorporates time as a retrieval dimension.
- **Semantic Representations**: Inverted indices typically work at the term level, while the structures described here operate at the semantic embedding level. This difference reflects the evolution from lexical to semantic retrieval.

B. Analysis of Practical Scenarios

To better understand potential benefits and limitations, we consider how these theoretical structures might address certain scenarios:

1) Concurrent Updates: When multiple updates occur simultaneously, maintaining consistency becomes important:

- **Temporal Ordering**: Temporal indexing could help ensure proper sequencing of updates, though this would require careful implementation to handle race conditions.
- **Atomicity Challenges**: Ensuring atomicity across updates to both vector stores and auxiliary structures would present significant implementation challenges.
- **Version Management**: Immutable data structures might help maintain consistent views, though this approach would trade off storage efficiency for consistency.

2) Embedding Evolution: As embedding models evolve or are retrained, vector representations of the same content can drift:

- **Model Versioning**: Metadata could track embedding model versions, which might help identify vectors that should be regenerated when models change.
- **Semantic Comparison**: Semantic signatures might help detect significant shifts in representation, though effective implementation of such signatures presents non-trivial challenges.
- **Selective Regeneration**: Lineage mapping could help identify affected vectors, though maintaining accurate lineage information at scale would require careful system design.

3) Retrieval Accuracy: Vector similarity search involves trade-offs between precision, recall, and performance:

- **Multi-stage Verification**: Combining vector similarity, metadata filtering, and semantic signatures could potentially provide multiple verification stages, though at the cost of increased computational complexity.
- **Context Filtering**: Metadata-based filtering could help narrow search results to relevant contexts, which might be implementable using existing vector database filtering capabilities.
- **Source Tracing**: When investigating results, lineage mapping could potentially help identify source documents, though similar functionality might be achievable through careful metadata management.

C. Cost-Benefit Considerations

Any additional system components introduce costs in terms of implementation, maintenance, and resources:

1) Storage Requirements: The described structures would require additional storage beyond the vector database itself:

- **Metadata Storage**: Storing metadata requires space proportional to the number of vectors and metadata fields, though much of this might be implementable within existing vector database capabilities.
- **Temporal Indices**: Maintaining temporal indexing would require additional storage, particularly if historical versions are preserved.
- **Lineage Information**: Storing document-vector relationships would require additional storage, which scales with the number of vectors and their relationships.
- **Semantic Signatures**: If implemented, semantic signatures would require additional storage for each vector.

2) Computational Considerations: The described structures would also introduce computational overhead:

- **Update Processing**: Each document update would require updating multiple data structures, potentially impacting ingestion performance.
- **Query Complexity**: Temporal and context-aware queries would involve additional filtering steps, potentially affecting query latency.

- **Signature Generation**: Computing semantic signatures would add processing time during document ingestion.

These costs would need to be weighed against potential benefits for specific use cases.

D. Potential Redundancies

Several aspects of the described structures might overlap with existing functionality:

- **Metadata Management**: Some metadata might duplicate information already present in document content or in vector database metadata.
- **Temporal Tracking**: Some aspects of temporal indexing might be addressable through existing version control or database timestamping.
- **Semantic Representations**: Semantic signatures might overlap with the information already captured in the embeddings themselves.

E. Summary

This analysis suggests that the auxiliary index structures described in this paper represent one potential approach to addressing certain operational considerations in RAG systems. Many of these considerations might also be addressable through existing technologies such as vector database metadata, version control systems, and information retrieval indices.

The value of implementing separate auxiliary structures would depend on specific requirements, constraints, and existing infrastructure in each deployment context. In some cases, leveraging and extending existing capabilities might be more efficient than implementing separate structures, while in others, dedicated auxiliary structures might offer benefits for particular use cases.

This theoretical exploration provides a framework for thinking about these trade-offs and considering how different approaches might address operational needs in RAG systems.

VIII. Related Work

This section reviews the foundational literature that directly informs our exploration. The concepts we adapt in this paper draw heavily from these established fields, and we acknowledge that our contribution lies primarily in synthesizing and applying these existing approaches to RAG systems rather than creating fundamentally new concepts.

A. Vector Databases and Indexing Techniques

Our understanding of vector search builds directly on the work of:

Johnson et al. [1], who introduced FAISS (Facebook AI Similarity Search), providing efficient implementations of approximate nearest neighbor search in high-dimensional spaces. Their work established core techniques for vector similarity search that we leverage in our exploration.

Malkov and Yashunin [2], who proposed the Hierarchical Navigable Small World (HNSW) algorithm, which forms the basis for many modern vector search implementations including those we discuss in this paper.

Commercial vector databases such as Pinecone [20], Milvus [21], and Weaviate [22] have extended basic vector indexing with metadata filtering capabilities that directly inform our metadata mapping approach. These systems demonstrate practical implementations of vector+metadata combinations that our exploration builds upon.

B. Retrieval-Augmented Generation

Our exploration of RAG systems extends directly from:

Lewis et al. [23], who introduced the original RAG architecture combining retrieval with generation. Their work established the foundation for the integration of retrieval and generation that we build upon.

Gao et al. [3], who presented a comprehensive survey of retrieval-augmented text generation methods. Their categorization helps contextualize where our exploration fits within the broader RAG ecosystem.

Asai et al. [24] proposed Self-RAG, which enhances RAG with self-reflection capabilities, allowing the model to decide when to retrieve and how to incorporate retrieved information. This approach improves the quality of generation through more intelligent retrieval integration.

Recent work by Kandpal et al. [4] and Wu et al. [5] has explored deployment aspects of RAG systems, addressing topics such as retrieval efficiency, result quality, and system reliability in practical applications.

C. Data Lineage and Provenance

Data lineage and provenance tracking have been extensively studied in database systems and data engineering, with applications to machine learning and AI systems [10], [11], [25].

Buneman et al. [10] established foundational concepts in data provenance, distinguishing between "where-provenance" (where data came from) and "why-provenance" (why data appears in the result). These concepts provide important frameworks for understanding the origins and influence of data.

Herschel et al. [11] provided a comprehensive survey of data provenance techniques, covering provenance capture, storage, and querying. Their taxonomy helps contextualize different approaches to tracking information lineage.

Miao et al. [25] explored provenance for machine learning, focusing on tracking the data and transformations that influence model predictions. Their work highlights the importance of provenance for debugging and explaining AI systems.

In the context of NLP and knowledge management, Chen et al. [12] and Xu et al. [13] have investigated hallucination detection and attribution in large language models. Their work examines approaches to tracing generated content back to source material for verification purposes.

D. Temporal Databases and Versioning Systems

Temporal database concepts and versioning systems provide important foundations for information management across time [6], [7], [14].

Jensen and Snodgrass [6] introduced temporal database concepts, including valid time (when facts are true in the real world) and transaction time (when facts are recorded in the database). These concepts enable point-in-time querying and historical analysis in database systems.

Tansel et al. [7] explored query languages and implementation techniques for temporal databases, providing insights into efficient storage and retrieval of time-varying data. Their work addresses important aspects of managing historical information.

Version control systems like Git [14] implement sophisticated mechanisms for tracking changes to documents over time. Git's branching model, commit history, and merge strategies provide powerful approaches to managing evolving content.

In the context of RAG systems, Park et al. [8] and Lin et al. [9] have explored temporal aspects of retrieval and evaluation, recognizing the importance of temporal consistency in knowledge-intensive applications. Their work examines approaches to handling time-dependent information in retrieval contexts.

E. Context-Aware Information Retrieval

Context-aware information retrieval spans from traditional search systems to modern AI-powered retrieval approaches [17], [18], [26], [27].

Ingwersen and Järvelin [26] proposed the "turn to context" in information retrieval, arguing that search systems should incorporate user context, task context, and domain context to improve result relevance. Their conceptual framework highlights the importance of contextual factors in search.

Dourish [27] explored context from a human-computer interaction perspective, distinguishing between representational approaches (context as information) and interactional approaches (context as emergent). This work provides valuable perspectives on how context can be conceptualized in information systems.

In the RAG domain, Yan et al. [17] and Gekhman et al. [18] have investigated multi-context and personalized retrieval for large language models. Their work examines approaches to context-sensitive retrieval in language model applications.

F. Semantic Analysis and Change Detection

Research on semantic analysis and change detection provides approaches to understanding meaning shifts in content [15], [16], [28].

Liu et al. [15] investigated the "lost in the middle" problem, where relevant information in the middle of retrieved documents is often overlooked by language models. Their findings highlight challenges in semantic processing of retrieved content.

Nakano et al. [16] developed WebGPT, which incorporates browsing and citation mechanisms to improve the factuality of language model outputs. Their approach explores evidence tracing and attribution in language model systems.

Work on semantic hashing [28] provides techniques for generating compact binary signatures that preserve semantic similarity relationships. These approaches offer efficient ways to represent and compare semantic content.

G. Enterprise Knowledge Management

Enterprise knowledge management systems have addressed various challenges in organizing and utilizing organizational knowledge [29]–[31].

Nonaka and Takeuchi [29] introduced the knowledge spiral model, emphasizing the conversion between tacit and explicit knowledge in organizations. Their perspective highlights the dynamic nature of knowledge in organizational contexts.

Davenport and Prusak [30] explored how organizations can effectively manage knowledge as a strategic asset, highlighting the importance of context, accessibility, and governance. Their work addresses practical aspects of knowledge management in enterprise settings.

Alavi and Leidner [31] provided a comprehensive review of knowledge management research, identifying key perspectives including knowledge as a state of mind, an object, a process, access to information, and a capability. Their taxonomy offers multiple frameworks for thinking about knowledge organization.

H. Summary

The literature reviewed in this section spans vector databases, RAG architectures, data lineage, temporal databases, context-aware retrieval, semantic analysis, and knowledge management. These diverse areas provide rich conceptual foundations for thinking about information organization in retrieval systems.

The theoretical exploration presented in the following sections draws inspiration from these existing approaches, considering how concepts from different domains might relate to auxiliary structures in information retrieval contexts. Rather than addressing gaps in existing work, this paper explores potential complementary perspectives on information organization for RAG systems.

IX. Conclusion

This paper has explored how established concepts from temporal databases, data provenance systems, and information retrieval might be adapted and combined to address emerging challenges in RAG systems. Our primary contribution lies not in creating novel mathematical structures, but in showing how existing approaches might be synthesized specifically for RAG applications.

A. Summary of Synthesis

Our exploration has drawn directly from several established fields:

- **Metadata Management**: We've examined how metadata approaches from information management systems might apply to vector retrieval, while acknowledging that modern vector databases already provide robust metadata capabilities that implement many of these concepts.
- **Temporal Database Concepts**: We've adapted well-established temporal database principles to the RAG

context, recognizing that these approaches have been thoroughly developed in database research over decades.
- **Data Provenance Techniques**: We've applied concepts from data lineage and provenance research to RAG systems, building on a rich tradition of work in database provenance tracking.
- **Context Isolation Methods**: We've drawn on information security and multi-tenant system concepts to consider context management in RAG, acknowledging the significant body of work in these areas.

B. Relationship to Existing Approaches

The concepts we've explored have substantial overlap with existing systems and approaches:

- Many vector databases already support metadata filtering that can address context management and temporal querying needs without requiring separate auxiliary structures.
- Version control systems like Git provide sophisticated mechanisms for tracking document changes that could be integrated with RAG systems to address many of the versioning challenges we've discussed.
- Data lineage systems in enterprise data pipelines already implement many of the provenance tracking capabilities we've explored.
- Application-level logic in many existing systems already implements similar capabilities to what we've described without requiring dedicated auxiliary structures.

C. Practical Implications

When considering how these concepts might be applied in practice:

- In many cases, extending existing vector databases with application-specific logic might be more practical than implementing separate auxiliary structures as we've described.
- Integration with existing version control and data provenance systems might provide more immediate benefits than developing new mathematical formulations.
- The choice of implementation approach should be guided by specific requirements and constraints rather than theoretical elegance.
- Many organizations might achieve similar benefits by carefully designing their RAG pipelines using existing tools rather than implementing new mathematical structures.

D. Future Directions

This exploration suggests several potential directions for those interested in RAG systems:

- Investigating how existing vector databases might be extended or configured to better support temporal and lineage tracking needs.
- Exploring integration patterns between vector databases and existing version control or information management systems.
- Examining how metadata and filtering capabilities in vector databases might be leveraged for specific application domains.
- Considering how lessons from temporal databases and data provenance might inform the design of RAG pipelines.

Despite the extensive literature in each individual domain, several research gaps remain at their intersection. The performance implications of maintaining comprehensive lineage information for high-volume RAG systems require further investigation. Additionally, formalizing the semantic consistency guarantees possible with these approaches remains an open question. While our mathematical formulation provides a starting point, empirical evaluation of these concepts in production environments would be valuable to assess their practical benefits and limitations.

E. Closing Thoughts

This paper has been an exploration of how established concepts from different domains might be synthesized to address emerging challenges in RAG systems. We've examined how mathematical formulations from temporal databases, data provenance, and information retrieval might be adapted to this context, while acknowledging the significant overlap with existing approaches.

Rather than claiming to introduce fundamentally new structures, we've attempted to show how existing concepts can be combined and applied to the specific challenges of RAG systems. We hope this synthesis contributes to the ongoing conversation about how to build more effective, trustworthy, and manageable RAG systems by drawing on the rich history of research across multiple disciplines.

References

[1] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535–547, 2019.

[2] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 4, pp. 824–836, 2018.

[3] Y. Gao, Y. Xiong, X. Gao, X. Han, J. Gao, Q. Si, and N. Duan, "Retrieval-augmented generation for large language models: A survey," *arXiv preprint arXiv:2312.10997*, 2023.

[4] N. Kandpal, X. Ren, Y. Li, T. Chen, K. Yuan, X. Wang, T. Zhao *et al.*, "Large language models, retrieval, and multi-step reasoning: A tutorial and survey," *arXiv preprint arXiv:2312.11613*, 2023.

[5] T. Wu, M. Terry, and C. J. Cai, "Ai chains: Transparent and controllable human-ai interaction by chaining large language model prompts," *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 2022.

[6] C. S. Jensen and R. T. Snodgrass, "Temporal data management," *IEEE Transactions on knowledge and data engineering*, vol. 11, no. 1, pp. 36–44, 1999.

[7] A. U. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, *Temporal databases: theory, design, and implementation*. Benjamin-Cummings Publishing Co., Inc., 2007.

[8] S. Park, J. Kim, and M. Seo, "Tart: A plug-and-play transformer module for task-agnostic reasoning," *arXiv preprint arXiv:2306.07536*, 2023.

[9] S. E. Lin, A. Desai, M. Sclar, I. Joshi, A. Deshpande, and C. Baral, "Ragas: Automated evaluation of retrieval augmented generation," *arXiv preprint arXiv:2309.15217*, 2023.

[10] P. Buneman, S. Khanna, and W.-C. Tan, "Why and where: A characterization of data provenance," *International conference on database theory*, pp. 316–330, 2001.

[11] M. Herschel, R. Diestelkämper, and H. Ben Lahmar, "A survey on provenance: What for? what form? what from?" *The VLDB Journal*, vol. 26, pp. 881–906, 2017.

[12] P. Chen, L. Qin, Y. Gu, X. Yao, H. Ji, and L. Bing, "Hallucination detection: Robustly identifying factual inconsistencies in large language models," *arXiv preprint arXiv:2311.01260*, 2023.

[13] Y. Xu, M. Phelps, W. Zhou, S. Cornell, S. Savarese, P. Liang, and J. Zou, "Large language models can be guided to evade ai-generated text detection," *arXiv preprint arXiv:2309.07384*, 2023.

[14] G. Project, "Git version control system," https://git-scm.com/, 2023.

[15] N. F. Liu, I. Beltagy, D. Downey, H. Hajishirzi, and D. Khashabi, "Lost in the middle: How language models use long contexts," *arXiv preprint arXiv:2307.03172*, 2023.

[16] R. Nakano, J. Hilton, S. Balaji, J. Wu, L. Ouyang, C. Kim, C. Hesse, S. Jain, V. Kosaraju, W. Saunders *et al.*, "Webgpt: Browser-assisted question-answering with human feedback," *arXiv preprint arXiv:2112.09332*, 2021.

[17] J. Yan, Y.-F. Wu, Z. Liu, and K. Zhou, "Multi-context retrieval augmented generation," *arXiv preprint arXiv:2310.16597*, 2023.

[18] Z. Gekhman, O. Cohen, V. Chu, I. Kung, O. Biton, O. Levy, and U. Alon, "Retrieval meets long context large language models," *arXiv preprint arXiv:2310.03025*, 2023.

[19] J. Guo, Y. Wang, D. Chen, J. Qiu, S. Ou, Z. Zhou, S. Lu, and J. Dai, "Vector databases: A survey," *arXiv preprint arXiv:2307.05582*, 2023.

[20] P. S. Inc., "Pinecone vector database," https://www.pinecone.io/, 2023.

[21] M. Team, "Milvus: An open source vector database for scalable similarity search," https://milvus.io/, 2021.

[22] W. Team, "Weaviate: A vector search engine and vector database," https://weaviate.io/, 2022.

[23] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.

[24] A. Asai, Z. Wu, Y. Wang, A. Sil, and H. Hajishirzi, "Self-rag: Learning to retrieve, generate, and critique through self-reflection," *arXiv preprint arXiv:2310.11511*, 2023.

[25] H. Miao, A. Li, L. S. Davis, and A. Deshpande, "Provenance-based interpretability for machine learning," *Proceedings of CIDR*, 2019.

[26] P. Ingwersen and K. Järvelin, "The turn to the user in information retrieval," *Journal of documentation*, vol. 61, no. 1, pp. 228–247, 2005.

[27] P. Dourish, "What we talk about when we talk about context," *Personal and ubiquitous computing*, vol. 8, pp. 19–30, 2004.

[28] J. Li, C. Liu, J. Yu, Y. Yang, C. Wang, T. Wang, X. Wang, and B. Wang, "Semantic hashing for nearest neighbor search," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 6, pp. 1160–1174, 2020.

[29] I. Nonaka and H. Takeuchi, "The knowledge-creating company: How japanese companies create the dynamics of innovation," *Oxford university press*, 1995.

[30] T. H. Davenport and L. Prusak, *Working knowledge: How organizations manage what they know*. Harvard Business Press, 1998.

[31] M. Alavi and D. E. Leidner, "Knowledge management and knowledge management systems: Conceptual foundations and research issues," *MIS quarterly*, pp. 107–136, 2001.

Auxiliary Index Structures for Contextualized.pdf

ORIGINALITY REPORT

9%	8%	7%	6%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	arxiv.org Internet Source	2%
2	www.biorxiv.org Internet Source	1%
3	d-nb.info Internet Source	1%
4	Submitted to University of York Student Paper	1%
5	Submitted to Higher Education Commission Pakistan Student Paper	<1%
6	Submitted to South Dakota Board of Regents Student Paper	<1%
7	Sasha Goldshtein, Dima Zurbalev, Ido Flatow. "Pro .NET Performance", Springer Nature, 2012 Publication	<1%
8	ShaowenAnandJames D.WenwuYong WangPadmanabhanMyersTangLiu. "Towards provenance-aware geographic information systems", Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems - GIS 08 GIS 08, 2008 Publication	<1%
9	www.ijcaonline.org Internet Source	<1%

10	J. Palmeri, P.M. Tuten. "Dialogic Negotiations: A Reflective Tale of Collaboration Across the Academic–Practitioner Divide", IEEE Transactions on Professional Communication, 2005 Publication	<1 %
11	Submitted to JIS University Student Paper	<1 %
12	neurips2023-enlsp.github.io Internet Source	<1 %
13	www.slideshare.net Internet Source	<1 %
14	sites.computer.org Internet Source	<1 %
15	Arnon Axelrod. "Complete Guide to Test Automation", Springer Science and Business Media LLC, 2018 Publication	<1 %
16	www.allbusiness.com Internet Source	<1 %
17	Submitted to Kingston University Student Paper	<1 %
18	www.ijraset.com Internet Source	<1 %
19	Submitted to University of Warwick Student Paper	<1 %
20	loyaltysurf.io Internet Source	<1 %
21	wiki.epfl.ch Internet Source	<1 %
22	esp.as-pub.com Internet Source	<1 %
23	www.pulumi.com Internet Source	

		<1%
24	cs.iit.edu Internet Source	<1%
25	export.arxiv.org Internet Source	<1%
26	iacat.org Internet Source	<1%
27	www.jove.com Internet Source	<1%
28	www.medialab.tfe.umu.se Internet Source	<1%
29	www.vbds.nl Internet Source	<1%
30	Zahra Khalila, Arbi Haza Nasution, Winda Monika, Aytug Onan, Yohei Murakami, Yasir Bin Ismail Radi, Noor Mohammad Osmani. "Investigating Retrieval-Augmented Generation in Quranic Studies: A Study of 13 Open-Source Large Language Models", International Journal of Advanced Computer Science and Applications, 2025 Publication	<1%
31	ir.cwi.nl Internet Source	<1%
32	"Intelligent Systems and Data Science", Springer Science and Business Media LLC, 2025 Publication	<1%
33	Xinyi Li, Sai Wang, Siqi Zeng, Yu Wu, Yi Yang. "A survey on LLM-based multi-agent systems: workflow, infrastructure, and challenges", Vicinagearth, 2024 Publication	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		