Access-Controlled Semantic Search.pdf

by Turnitin Student

Submission date: 22-Jun-2025 06:48PM (UTC-0500)
Submission ID: 2703542407
File name: Access-Controlled_Semantic_Search.pdf (395.4K)
Word count: 11136
Character count: 78121

Access-Controlled Semantic Search: Implementing Role-Based Filtering in Vector Databases for Enterprise Document Management

Mateus Yonathan
Software Developer & Independent Researcher

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

***Abstract*—Vector databases used for semantic search in enterprise environments lack integrated access control mechanisms, creating security gaps when deploying retrieval-augmented generation (RAG) systems. This paper presents a technical approach for implementing role-based access control by embedding access metadata directly in vector database payloads. We formalize access-controlled vector operations and demonstrate their conceptual implementation using Qdrant's payload filtering capabilities with .NET 9. Our approach enables atomic operations where access control decisions and semantic similarity rankings occur within the same database transaction. The conceptual framework demonstrates feasibility of fine-grained access control for semantic search without requiring separate security systems or application-layer filtering. We position this as an engineering contribution that applies existing database filtering capabilities to enterprise document management rather than a fundamental algorithmic innovation.**

I. Introduction

Software engineers building enterprise RAG systems face a critical implementation challenge: how to integrate role-based access control with vector databases while maintaining both security and performance. Most vector database tutorials and documentation focus on basic similarity search, ignoring the access control requirements that are essential for production enterprise deployment.

Consider a typical enterprise scenario: your development team has successfully implemented a semantic search system using Qdrant and .NET, but when security reviews the system for production deployment, they discover that any authenticated user can potentially access any document chunk through semantic similarity queries. The vector database treats all embeddings as equally accessible, regardless of the original document's access restrictions. This challenge is particularly acute given the scale of modern vector databases [1] and the efficiency of approximate nearest neighbor search algorithms [2].

This creates several engineering challenges that development teams must solve before production deployment:

- **Access Control Integration**: How to enforce existing organizational permissions within vector database queries
- **Audit Requirements**: How to track which documents contributed to AI responses for compliance reporting
- **Performance Constraints**: How to implement access control without degrading search performance
- **Consistency Guarantees**: How to maintain synchronization between access policies and vector data during updates

Current implementation approaches have significant limitations for enterprise development:

Application-Layer Filtering: Implementing access control in application code after vector search results in security vulnerabilities and performance issues. Users can potentially access unauthorized content through carefully crafted queries. This approach violates fundamental access control principles [3].

Separate Authorization Services: Using external authorization services creates consistency challenges and additional network latency. The authorization service cannot understand vector similarity context, creating gaps in enterprise security architectures [4].

Document-Level Permissions: Traditional document management systems provide coarse-grained access control but cannot handle chunk-level permissions required for semantic search applications. Classical information retrieval systems [5] were not designed for the fine-grained access control requirements of modern AI applications.

This paper provides a practical architectural framework for software engineers that leverages Qdrant's payload filtering capabilities to implement fine-grained access control directly in the vector database. Our engineering contributions include:

1) **Formal Model**: Mathematical definitions for access-controlled vector operations that maintain consistency between permissions and semantic rankings
2) **Implementation Pattern**: A .NET-based integration approach using Qdrant's payload filtering capabilities to enforce role-based access control at query time
3) **Consistency Guarantees**: Protocols for maintaining metadata-vector synchronization during concurrent document updates

We explicitly scope this work to PDF document management in enterprise environments where documents have well-defined access control requirements. We do not claim to solve general AI governance or provide novel vector database algorithms.

Our approach recognizes that enterprise document archives are not simply collections of text to be processed, but repositories of institutional knowledge with complex relationships, varying authority levels, and strict access requirements. The framework we present addresses these challenges through three core innovations: chunk-level role enforcement that maintains access controls even after document processing, semantic lineage tracking that preserves the relationship between generated answers and source materials, and autonomous content governance that identifies and addresses inconsistencies, outdated information, and policy violations without human intervention.

The conceptual system we describe operates on principles borrowed from database management, information security, and compliance frameworks, adapted specifically for the unique challenges of AI-powered document interaction. By maintaining local inference capabilities and zero-trust architectures, the system ensures that sensitive information never leaves organizational boundaries while still providing the natural language capabilities that make AI valuable for knowledge work.

This work contributes a formal framework for understanding how enterprise AI systems can maintain the trust and compliance requirements necessary for deployment in regulated environments while still delivering the user experience benefits that drive AI adoption. Rather than presenting a deployed system, we offer a conceptual foundation that system architects can adapt to their specific organizational needs and compliance requirements.

The remainder of this paper develops this framework through formal mathematical models, architectural specifications, and governance mechanisms that collectively address the question of how organizations can safely transform their PDF archives from information graveyards into governed knowledge assets. We begin by establishing a system model that captures the unique requirements of enterprise document governance, then proceed to develop the mathematical foundations necessary for implementing these concepts in practice.

II. System Model

Our confidential PDF knowledge governance layer operates on a fundamentally different model than traditional RAG systems. Where conventional approaches treat documents as uniform inputs to be processed and retrieved, our system recognizes that enterprise documents exist within complex organizational contexts that must be preserved throughout the AI interaction lifecycle.

A. Core Components

The system consists of five primary components working in concert to maintain governance while enabling AI-powered document interaction:

Document Ingestion Pipeline: This component handles the initial processing of PDF documents, extracting both content and metadata while preserving organizational context. Unlike simple PDF parsers, this pipeline maintains document provenance, version history, and access classification throughout the ingestion process. The pipeline performs semantic chunking that respects document structure and organizational boundaries, ensuring that chunks maintain meaningful relationships to their source documents and organizational context.

Chunk Governance Layer: Each semantic chunk carries metadata that includes role-based access controls, temporal validity, and relationship mappings to other chunks and documents. This layer implements fine-grained access enforcement that operates at the individual chunk level rather than the document level, allowing for more precise control over information exposure while maintaining semantic coherence.

Vector Database with Access Payloads: Built on Qdrant, this component stores semantic embeddings alongside governance metadata. Unlike traditional vector databases that focus purely on similarity search, our approach embeds access control information directly into the database payloads, enabling access-aware similarity searches that respect organizational permissions and roles.

Contextual Query Engine: This component processes natural language queries while maintaining awareness of user roles, organizational policies, and access constraints. Rather than simply finding semantically similar content, the query engine ensures that all retrieved information respects access controls and maintains audit trails for compliance purposes.

Autonomous Governance Agents: These agents continuously monitor the document corpus for inconsistencies, outdated information, and policy violations. They operate independently of user queries to maintain document quality and compliance, identifying when information becomes outdated, conflicts arise between documents, or access patterns suggest potential security concerns.

B. Operational Principles

The system operates under several key principles that distinguish it from conventional document management and RAG approaches:

Zero-Trust by Default: All access decisions are made explicitly based on current user roles and document classifications. No information is ever served without explicit authorization, and all authorization decisions are logged for audit purposes. This principle extends to system components, where no component trusts any other component's access decisions without verification.

Semantic Preservation: Document chunking and processing preserve the semantic relationships that exist within and between documents. This ensures that AI-generated responses maintain the context and authority relationships that exist in the original document structure, preventing the system from creating artificial connections between unrelated information.

Temporal Awareness: The system maintains awareness of when information was created, modified, and last validated. This temporal dimension enables the system to prioritize current information, identify potentially outdated content,

and maintain historical views of document evolution for compliance purposes.

Local Inference Only: All AI processing occurs within organizational boundaries using local language models. This principle ensures that sensitive information never leaves the organization while still providing the natural language capabilities that make AI valuable for knowledge work.

C. Trust Model

Our trust model assumes that organizational boundaries represent the primary security perimeter, with fine-grained access controls operating within that perimeter. The system trusts user authentication mechanisms provided by organizational identity providers but makes all access decisions independently based on document classifications and user roles.

The model explicitly does not trust document content to be correctly classified or up-to-date without verification. Autonomous governance agents continuously validate content classifications and identify inconsistencies that require attention. This approach recognizes that enterprise document repositories often contain information with varying levels of authority and accuracy, requiring active management rather than passive storage.

D. Compliance Integration

The system model includes built-in compliance mechanisms that generate audit trails, maintain data lineage, and support regulatory reporting requirements. These mechanisms operate continuously rather than as post-hoc additions, ensuring that compliance information is available for any interaction with the system.

The compliance model addresses common enterprise requirements including data retention policies, access logging, and change management procedures. By integrating these requirements into the core system model rather than treating them as additional features, the system ensures that compliance considerations are addressed consistently across all operations.

This system model provides the foundation for the mathematical formalization that follows, where we develop formal definitions for the access control mechanisms, semantic preservation requirements, and governance procedures that enable safe AI adoption in enterprise environments.

III. Mathematical Model

This section formalizes the mathematical properties required for access-controlled vector operations in document management systems. We focus specifically on operational semantics that can be implemented using existing vector database filtering capabilities.

A. System State Definition

Let $\mathcal{D} = \{d_1, d_2, ..., d_n\}$ be a finite set of PDF documents and $\mathcal{R} = \{r_1, r_2, ..., r_k\}$ be a finite set of organizational roles with total order $r_1 < r_2 < ... < r_k$ where $r_i < r_j$ means role r_j has strictly greater access privileges than role r_i.

Definition 1 (Document State): Each document $d_i \in \mathcal{D}$ is a tuple $(content_i, metadata_i, timestamp_i)$ where:

- $content_i \in \Sigma^*$ is the document's textual content
- $metadata_i = (minRole_i, authorityLevel_i)$ with $minRole_i \in \mathcal{R}$ and $authorityLevel_i \in \{1, 2, 3, 4\}$
- $timestamp_i \in \mathbb{N}$ represents the last modification time

Definition 2 (Chunking Function): For document d_i, the deterministic chunking function $\phi : \mathcal{D} \to 2^{\mathcal{C}}$ produces a finite set of chunks:

$$\phi(d_i) = \{c_{i,1}, c_{i,2}, ..., c_{i,m_i}\}$$

where each chunk $c_{i,j} = (text_{i,j}, vector_{i,j}, acl_{i,j}, parent_i)$ with:

- $text_{i,j} \in \Sigma^*$ is the chunk's textual content
- $vector_{i,j} \in \mathbb{R}^d$ is the embedding vector
- $acl_{i,j} = (minRole_{i,j}, expires_{i,j})$ with $minRole_{i,j} \geq minRole_i$
- $parent_i$ is a reference to the source document

B. Access Control Formalization

Definition 3 (Access Control Predicate): For user with role $r_u \in \mathcal{R}$ at time $t \in \mathbb{N}$, chunk $c_{i,j}$ is accessible iff:

$$\text{accessible}(c_{i,j}, r_u, t) = (r_u \geq minRole_{i,j}) \wedge (t \leq expires_{i,j}) \quad (1)$$

Definition 4 (Filtered Vector Search): Given query vector $q \in \mathbb{R}^d$, user role r_u, time t, and limit $k \in \mathbb{N}$, the access-controlled retrieval operation returns:

$$\text{search}(q, r_u, t, k) = \text{top-k}(S_{accessible}, \text{sim}(q, \cdot)) \quad (2)$$

where $S_{accessible} = \{c_{i,j} : \text{accessible}(c_{i,j}, r_u, t)\}$ and sim denotes cosine similarity.

This formulation directly maps to Qdrant's filtered search capabilities where access control metadata is stored in the payload and filtering is applied before similarity ranking. This approach builds on fundamental database principles [6] by treating access control as a first-class database constraint rather than an application-layer concern.

C. Consistency Guarantees

Definition 5 (System State): At any time t, the system state is $\Sigma_t = (\mathcal{C}_t, \mathcal{M}_t)$ where $\mathcal{C}_t$ is the set of all chunks and $\mathcal{M}_t$ is the metadata mapping function such that $\mathcal{M}_t(c_{i,j}) = acl_{i,j}$.

Definition 6 (Atomic Update): An update operation $\text{update}(d_i, content', r_u, t)$ is atomic iff it produces a new system state Σ_{t+1} where:

1) All chunks $c_{i,j} \in \phi(d_i)$ are removed from $\mathcal{C}_t$
2) New chunks $c'_{i,j} \in \phi(d'_i)$ are added to $\mathcal{C}_{t+1}$ where $d'_i = (content', metadata_i, t + 1)$
3) No intermediate state exists where some old chunks remain while new chunks are partially added

This ensures that access control decisions are never made on inconsistent metadata during concurrent document updates.

D. Audit Operations

Definition 7 (Audit Record): Each system operation generates an immutable audit record:

$$\text{audit}(op, r_u, t, I, O) = (op, r_u, t, h(I), h(O))$$

where op is the operation type, r_u is the user role, t is the timestamp, I and O are inputs and outputs, and h is a cryptographic hash function.

Definition 8 (Audit Trail Completeness): The audit trail $\mathcal{A}_t$ at time t contains exactly one record for each state transition in $\{\Sigma_0, \Sigma_1, ..., \Sigma_t\}$, enabling complete system state reconstruction.

This mathematical framework provides the formal foundation for implementing access-controlled vector operations using standard database filtering capabilities, with clear operational semantics that can be directly translated to working code.

The mathematical model provides formal guarantees about access control enforcement, semantic preservation, and audit completeness that are essential for enterprise deployment. Implementation of this model requires careful attention to computational complexity, particularly for large document corpora, but the conceptual foundation ensures that governance requirements can be met while maintaining the performance characteristics necessary for practical AI applications.

This formalization serves as the specification for the architectural components described in the following section, where we translate these mathematical concepts into practical system designs using .NET 9 and modern vector database technologies.

IV. System Architecture

The access-controlled semantic search system implements a layered architecture that integrates role-based access control directly into vector database operations. This section describes the core components and their interactions.

A. Core Components

Document Processing Pipeline: The document ingestion pipeline processes PDF files through chunking, embedding generation, and metadata extraction. Each document chunk receives governance metadata that encodes access control requirements directly into the vector database payload.

Vector Database Layer: The system uses Qdrant as the primary vector database, extended with custom payload structures that embed governance metadata directly into vector database records. This approach ensures that similarity searches automatically respect role-based access control without requiring external filtering.

Access Control Service: A dedicated service manages role hierarchies, temporal permissions, and audit logging. This service integrates with existing enterprise identity systems and provides real-time access control decisions for vector database queries.

Query Processing Engine: The query processor transforms natural language queries into vector searches while applying appropriate access control filters. The engine ensures that only authorized content contributes to response generation.

B. Integration Patterns

The architecture implements several key integration patterns that enable enterprise deployment:

Identity System Integration: The framework integrates with Active Directory, LDAP, and modern identity providers through standard protocols (SAML, OAuth, OpenID Connect). Role mappings are synchronized in real-time to ensure access control decisions reflect current organizational structure. This integration follows enterprise architecture patterns [4] for distributed system design.

Audit Trail Generation: Every system operation generates structured audit events that capture who accessed what information, when, and why. These audit trails support compliance reporting and security analysis while providing transparency into AI decision-making processes.

Response Generation: The language model generates natural language responses based on the assembled context, with prompts designed to preserve source attribution and indicate confidence levels. Response generation operates under the same access control constraints as the underlying vector search.

C. Deployment Architecture

The system supports multiple deployment patterns to accommodate different enterprise requirements:

On-Premises Deployment: Complete system deployment within enterprise infrastructure, supporting air-gapped environments and sensitive data processing requirements.

Hybrid Cloud: Selective cloud deployment of compute-intensive components while maintaining sensitive data on-premises.

Multi-Tenant SaaS: Cloud-native deployment with tenant isolation and shared infrastructure for organizations requiring managed services. This deployment pattern leverages microservices architecture principles [7] to ensure scalability and maintainability.

V. Governance Framework

The governance framework translates the mathematical model and system architecture into operational procedures that ensure ongoing compliance and system integrity. This framework addresses the practical challenges of maintaining document governance in dynamic enterprise environments where policies change, organizational structures evolve, and compliance requirements shift.

A. Role-Based Access Management

The framework implements a sophisticated role-based access control system that extends beyond simple user permissions to consider contextual factors such as time, location, and purpose of access. Unlike traditional systems that apply access controls at the document level, our framework operates at the chunk level, enabling fine-grained information sharing that respects both security requirements and business needs.

Role definitions in the system capture not just hierarchical relationships but also functional responsibilities. For example, an HR representative might have access to personnel policies regardless of their position in the organizational hierarchy, while a senior executive might be restricted from accessing detailed financial audit data despite their overall authority level. This nuanced approach to access control reflects the complex information sharing patterns that exist in real organizations [8].

The system maintains dynamic role mappings that automatically adjust access permissions as organizational structures change. When employees change roles, receive promotions, or move between departments, their access permissions update automatically based on predefined governance policies. This automation reduces the administrative burden of access management while ensuring that access controls remain current and appropriate.

B. Document Lifecycle Management

The governance framework implements comprehensive document lifecycle management that addresses the full spectrum of document states from creation through archival. This lifecycle management extends beyond traditional document management systems by maintaining semantic coherence and governance metadata throughout all state transitions.

Creation and Classification: When documents enter the system, they undergo automated classification that determines initial access controls, retention periods, and governance requirements. This classification process considers document content, source context, and organizational policies to assign appropriate governance metadata without requiring manual intervention for routine documents.

Review and Validation: The framework implements periodic review processes that ensure documents remain current and accurate. These reviews consider factors such as regulatory changes, organizational policy updates, and usage patterns to identify documents that require attention. The review process operates through a combination of automated analysis and human oversight, escalating documents that require expert review while handling routine validations automatically.

Version Control and Lineage: Every document change is tracked through comprehensive version control that maintains semantic relationships between versions. This version control system enables temporal queries that can access information as it existed at specific points in time, crucial for compliance scenarios where organizations need to demonstrate historical decision-making processes.

Archival and Retention: The framework implements sophisticated archival procedures that respect both legal retention requirements and operational efficiency concerns. Documents transition through various retention states, from active use through archival storage, while maintaining searchability and compliance capabilities throughout their lifecycle.

C. Compliance Integration

The governance framework integrates compliance requirements directly into operational procedures rather than treating compliance as an afterthought. This integration ensures that compliance considerations are addressed consistently across all system operations.

Regulatory Alignment: The framework maps regulatory requirements to specific governance procedures, ensuring that document handling procedures automatically comply with applicable regulations. This mapping is maintained through configuration rather than code changes, enabling organizations to adapt to regulatory changes without system modifications.

Audit Trail Management: Every interaction with the system generates audit records that support compliance reporting and legal discovery processes. These audit trails capture not just what information was accessed but also the decision-making process that led to access approval, providing the documentation necessary for regulatory compliance.

Privacy Protection: The framework implements privacy protection measures that go beyond access control to include data minimization, purpose limitation, and consent management. These measures ensure that personal information is handled appropriately throughout the document lifecycle.

D. Quality Assurance Procedures

The governance framework includes comprehensive quality assurance procedures that maintain information accuracy and reliability. These procedures address the common enterprise challenge of ensuring that AI systems provide accurate, current, and trustworthy information.

Content Validation: Automated content validation procedures identify inconsistencies, contradictions, and potential errors in the document corpus. These procedures consider not just individual document content but also relationships between documents, identifying cases where different documents provide conflicting information on the same topic.

Currency Monitoring: The framework monitors document currency through automated analysis of content, metadata, and usage patterns. Documents that may be outdated are flagged for review, ensuring that AI responses are based on current and accurate information.

Source Authority Tracking: The system maintains awareness of document authority levels, ensuring that information from authoritative sources takes precedence over less reliable information. This authority tracking helps prevent situations where AI systems provide answers based on draft documents, personal opinions, or unofficial communications.

E. Incident Response Procedures

The governance framework includes comprehensive incident response procedures that address the unique challenges of AI-powered document systems. These procedures ensure that organizations can respond effectively to security incidents, compliance violations, or system failures while maintaining audit trail integrity.

Access Violation Response: When unauthorized access attempts are detected, the system implements automated response procedures that isolate potentially compromised

information while preserving audit evidence. These procedures balance security concerns with operational continuity, ensuring that legitimate users can continue to access authorized information while suspected violations are investigated.

Data Quality Incidents: When content quality issues are identified, the framework implements procedures for rapid response that can temporarily restrict access to questionable information while investigations proceed. These procedures ensure that users are not provided with potentially incorrect information while maintaining transparency about system limitations.

Compliance Violations: The framework includes procedures for responding to potential compliance violations, including automated notifications to compliance officers, preservation of relevant audit information, and coordination with legal and regulatory response teams.

F. Continuous Improvement Processes

The governance framework implements continuous improvement processes that adapt governance procedures based on operational experience, changing requirements, and emerging best practices.

Performance Monitoring: The framework continuously monitors governance effectiveness through metrics that track access control accuracy, document currency, user satisfaction, and compliance posture. These metrics provide the foundation for ongoing governance improvements.

Policy Evolution: Governance policies are treated as living documents that evolve based on operational experience and changing requirements. The framework includes procedures for policy updates that ensure changes are implemented consistently across the system while maintaining audit trail integrity [9].

User Feedback Integration: The framework incorporates user feedback into governance improvements, recognizing that effective governance must balance security requirements with user productivity. This feedback helps identify areas where governance procedures can be streamlined without compromising security or compliance requirements.

This governance framework provides the operational foundation necessary for enterprise deployment of confidential PDF knowledge systems. The framework ensures that the conceptual guarantees provided by the mathematical model are realized in practice through comprehensive operational procedures that address the full spectrum of enterprise governance requirements.

VI. Implementation Design

This section provides technical specifications for implementing the confidential PDF knowledge governance layer using .NET 9, focusing on the specific language features and architectural patterns that enable enterprise-grade governance capabilities while maintaining performance and maintainability.

A. Core Data Structures

The implementation leverages .NET 9's enhanced record types [10] and immutable collections to ensure data integrity throughout the governance pipeline. The core data structures implement the mathematical model's entities while providing practical interfaces for enterprise integration.

Document Representation: Documents are represented using .NET 9's immutable record types that capture both content and governance metadata. The immutable nature of these records ensures that document state changes are explicit and auditable, supporting the lineage tracking requirements defined in our mathematical model.

Listing 1: Document Governance Record Structure

```
public sealed record DocumentRecord(
    Guid DocumentId,
    string Title,
    string ContentHash,
    AuthorityLevel Authority,
    DateTimeOffset CreatedAt,
    DateTimeOffset LastModified,
    ImmutableList<AccessRole> RequiredRoles,
    DocumentLineage Lineage
);

public sealed record ChunkRecord(
    Guid ChunkId,
    Guid ParentDocumentId,
    string Content,
    ReadOnlyMemory<float> Embedding,
    AccessRole MinimumRole,
    DateTimeOffset ExpiresAt,
    ChunkMetadata Metadata
);
```

Chunk Governance Records: Each semantic chunk is represented by an immutable record that includes content, embedding vectors, access control metadata, and lineage information. These records implement the chunk-level access control requirements defined in the mathematical model while providing efficient serialization for vector database storage.

Temporal Lineage Structures: Document evolution is tracked through immutable lineage records that form directed acyclic graphs representing document derivation relationships. These structures enable efficient temporal queries while maintaining complete audit trails.

Access Control Contexts: User access contexts are represented through structured records that capture current roles, time bounds, and access history. These contexts provide the input for access control decisions while supporting audit requirements:

Listing 2: Access Control Implementation

```
public sealed class AccessControlService
{
    public async Task<AccessDecision>
        EvaluateAccessAsync(
        ChunkRecord chunk,
        UserContext user,
```

```
        CancellationToken cancellationToken =
            default)
    {
        var hasRole = user.Roles.Any(role =>
            role.Authority >= chunk.MinimumRole.
                Authority);

        var isActive = chunk.ExpiresAt >
            DateTimeOffset.UtcNow;

        var decision = new AccessDecision(
            IsGranted: hasRole && isActive,
            User: user.UserId,
            Chunk: chunk.ChunkId,
            DecisionTime: DateTimeOffset.UtcNow,
            Reason: hasRole ? "Authorized" : "
                Insufficient role"
        );

        await _auditLogger.LogAccessDecisionAsync(
            decision, cancellationToken);
        return decision;
    }
}
```

B. Concurrency and Threading Model

The implementation uses .NET 9's enhanced asynchronous programming capabilities to handle the concurrent access patterns typical of enterprise knowledge management systems. The threading model prioritizes consistency for governance operations while optimizing performance for query operations.

Governance Operation Synchronization: Document governance operations use distributed locks implemented through consensus algorithms to ensure consistency across multiple service instances. This approach ensures that access control decisions remain consistent even during high-concurrency scenarios.

Query Operation Optimization: Query operations use parallel processing capabilities to optimize performance while maintaining access control enforcement. The implementation leverages .NET 9's parallel LINQ extensions to distribute similarity searches across available processing cores while respecting access boundaries.

Background Governance Processing: Autonomous governance agents operate through background services that use .NET 9's hosted service framework. These services implement work distribution patterns that ensure governance operations continue even during system maintenance or partial failures:

Listing 3: Governance Agent Background Service

```
public sealed class GovernanceAgentService :
    BackgroundService
{
    private readonly IContentValidator _validator;
    private readonly IDocumentRepository
        _repository;
    private readonly INotificationService
        _notifications;
    private readonly ILogger<GovernanceAgentService
        > _logger;
    private readonly IOptions<GovernanceOptions>
        _options;

    public GovernanceAgentService(
        IContentValidator validator,
        IDocumentRepository repository,
        INotificationService notifications,
        ILogger<GovernanceAgentService> logger,
        IOptions<GovernanceOptions> options)
    {
        _validator = validator;
        _repository = repository;
        _notifications = notifications;
        _logger = logger;
        _options = options;
    }

    protected override async Task ExecuteAsync(
        CancellationToken stoppingToken)
    {
        var timer = new PeriodicTimer(TimeSpan.
            FromMinutes(_options.Value.
            ValidationIntervalMinutes));

        while (await timer.WaitForNextTickAsync(
            stoppingToken))
        {
            await ValidateDocumentCorpusAsync(
                stoppingToken);
            await CleanupExpiredChunksAsync(
                stoppingToken);
            await UpdateDocumentCurrencyAsync(
                stoppingToken);
        }
    }

    private async Task ValidateDocumentCorpusAsync(
        CancellationToken cancellationToken)
    {
        await foreach (var batch in _repository.
            GetDocumentBatchesAsync(
            batchSize: _options.Value.BatchSize,
                cancellationToken))
        {
            var validationTasks = batch.Select(
                async doc =>
                {
                    try
                    {
                        var issues = await _validator.
                            ValidateContentAsync(doc,
                            cancellationToken);
                        if (issues.Any())
                        {
                            await _notifications.
                                NotifyContentOwnerAsync
                                (
                                doc.OwnerId, doc.
                                    DocumentId, issues
                                    ,
                                    cancellationToken)
                                    ;

                            _logger.LogWarning("
                                Document {DocumentId}
                                has {IssueCount}
                                validation issues",
                                doc.DocumentId, issues.
                                    Count());
                        }
                    }
                    catch (Exception ex)
                    {
                        _logger.LogError(ex, "Failed to
                             validate document {
                            DocumentId}", doc.
```

```
                    DocumentId);
            }
        });

        await Task.WhenAll(validationTasks);
    }
}
```

C. Vector Database Integration

The system integrates deeply with Qdrant to provide access-aware vector search capabilities. The integration maintains governance metadata alongside embeddings while optimizing for search performance:

Listing 4: Qdrant Vector Service Implementation

```
public sealed class QdrantVectorService : IVectorService
{
    private readonly QdrantClient _client;
    private readonly IAccessControlService _accessControl;
    private readonly ILogger<QdrantVectorService> _logger;
    private readonly IOptions<QdrantOptions> _options;

    public QdrantVectorService(
        QdrantClient client,
        IAccessControlService accessControl,
        ILogger<QdrantVectorService> logger,
        IOptions<QdrantOptions> options)
    {
        _client = client;
        _accessControl = accessControl;
        _logger = logger;
        _options = options;
    }

    public async Task<Guid> StoreChunkAsync(
        ChunkRecord chunk,
        UserContext user,
        CancellationToken cancellationToken = default)
    {
        var accessDecision = await _accessControl.EvaluateAccessAsync(chunk, user, cancellationToken);
        if (!accessDecision.IsGranted)
        {
            throw new UnauthorizedAccessException($"User {user.UserId} cannot store chunk {chunk.ChunkId}");
        }

        var pointId = Guid.NewGuid();
        var payload = new Dictionary<string, object>
        {
            ["chunk_id"] = chunk.ChunkId.ToString(),
            ["document_id"] = chunk.ParentDocumentId.ToString(),
            ["content"] = chunk.Content,
            ["minimum_role"] = chunk.MinimumRole.ToString(),
            ["expires_at"] = chunk.ExpiresAt.ToUnixTimeSeconds(),
            ["created_by"] = user.UserId.ToString(),
            ["authority_level"] = chunk.Metadata.AuthorityLevel,
            ["content_hash"] = chunk.Metadata.ContentHash,
            ["lineage"] = JsonSerializer.Serialize(chunk.Metadata.Lineage)
        };

        var point = new PointStruct
        {
            Id = pointId,
            Vectors = chunk.Embedding.ToArray(),
            Payload = payload
        };

        await _client.UpsertAsync(_options.Value.CollectionName, [point], cancellationToken);

        _logger.LogInformation("Stored chunk {ChunkId} as point {PointId} for user {UserId}", chunk.ChunkId, pointId, user.UserId);

        return pointId;
    }

    public async Task<IEnumerable<ChunkSearchResult>> SearchAsync(
        ReadOnlyMemory<float> queryVector,
        UserContext user,
        int limit = 10,
        CancellationToken cancellationToken = default)
    {
        var userRoles = user.Roles.Select(r => r.ToString()).ToList();
        var currentTime = DateTimeOffset.UtcNow.ToUnixTimeSeconds();

        var filter = new Filter
        {
            Must = new List<Condition>
            {
                new()
                {
                    Field = "minimum_role",
                    Match = new MatchAny { Any = userRoles }
                },
                new()
                {
                    Field = "expires_at",
                    Range = new Range { Gte = currentTime }
                }
            }
        };

        var searchRequest = new SearchPoints
        {
            CollectionName = _options.Value.CollectionName,
            Vector = queryVector.ToArray(),
            Limit = (uint)limit,
            Filter = filter,
            WithPayload = true,
            WithVector = false
        };

        var response = await _client.SearchAsync(searchRequest, cancellationToken);
```

```
        var results = response.Result.Select(point
            => new ChunkSearchResult
        {
            ChunkId = Guid.Parse(point.Payload["
                chunk_id"].StringValue),
            DocumentId = Guid.Parse(point.Payload["
                document_id"].StringValue),
            Content = point.Payload["content"].
                StringValue,
            Score = point.Score,
            AuthorityLevel = point.Payload["
                authority_level"].IntegerValue,
            ContentHash = point.Payload["
                content_hash"].StringValue
        }).ToList();

        _logger.LogInformation("Found {ResultCount}
             accessible chunks for user {UserId}",
            results.Count, user.UserId);

        return results;
    }
}
```

D. Persistence and Serialization

The implementation uses a multi-tier persistence strategy that optimizes for different data access patterns while maintaining governance requirements across all storage layers.

Governance Metadata Storage: Critical governance metadata is stored using transactional databases that provide ACID guarantees for access control decisions. The implementation uses Entity Framework Core with custom value converters to serialize complex governance structures while maintaining query performance.

Vector Database Integration: Semantic embeddings and associated governance payloads are stored in Qdrant using JSON serialization optimized for query performance. Custom serialization contracts ensure that governance metadata is efficiently encoded while maintaining full fidelity for access control decisions.

Audit Log Persistence: Audit logs use append-only storage patterns that provide tamper-evident audit trails. The implementation leverages .NET 9's System.Text.Json capabilities to serialize audit events with minimal overhead while maintaining comprehensive audit coverage.

E. API Controller Implementation

The system exposes RESTful APIs that maintain governance requirements while providing intuitive interfaces for document queries and management:

Listing 5: Knowledge Query API Controller

```
[ApiController]
[Route("api/[controller]")]
[Authorize]
public sealed class KnowledgeController :
    ControllerBase
{
    private readonly IKnowledgeQueryService
        _queryService;
    private readonly IDocumentIngestionService
        _ingestionService;
    private readonly IUserContextService
        _userContext;
    private readonly IAuditLogger _auditLogger;
    private readonly ILogger<KnowledgeController>
        _logger;

    public KnowledgeController(
        IKnowledgeQueryService queryService,
        IDocumentIngestionService ingestionService,
        IUserContextService userContext,
        IAuditLogger auditLogger,
        ILogger<KnowledgeController> logger)
    {
        _queryService = queryService;
        _ingestionService = ingestionService;
        _userContext = userContext;
        _auditLogger = auditLogger;
        _logger = logger;
    }

    [HttpPost("query")]
    [ProducesResponseType(typeof(QueryResponse),
        StatusCodes.Status200OK)]
    [ProducesResponseType(typeof(ProblemDetails),
        StatusCodes.Status400BadRequest)]
    [ProducesResponseType(typeof(ProblemDetails),
        StatusCodes.Status403Forbidden)]
    public async Task<ActionResult<QueryResponse>>
        QueryAsync(
        [FromBody] QueryRequest request,
        CancellationToken cancellationToken =
            default)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }

        var user = await _userContext.
            GetCurrentUserAsync(User,
            cancellationToken);
        var queryId = Guid.NewGuid();

        await _auditLogger.LogQueryAttemptAsync(
            queryId, user.UserId, request.Query,
            cancellationToken);

        try
        {
            var response = await _queryService.
                ProcessQueryAsync(
                request.Query, user, queryId,
                    cancellationToken);

            await _auditLogger.LogQuerySuccessAsync
                (
                queryId, user.UserId, response.
                    SourceChunks.Count(),
                    cancellationToken);

            _logger.LogInformation("Query {QueryId}
                 processed successfully for user {
                UserId}, " +
                "returned {ChunkCount} chunks",
                    queryId, user.UserId, response
                    .SourceChunks.Count());

            return Ok(response);
        }
        catch (UnauthorizedAccessException ex)
        {
            await _auditLogger.LogQueryDeniedAsync(
```

```
                queryId, user.UserId, ex.Message,
                cancellationToken);

            _logger.LogWarning("Query {QueryId}
                denied for user {UserId}: {Reason
                }",
                queryId, user.UserId, ex.Message);

            return Forbid();
        }
        catch (Exception ex)
        {
            await _auditLogger.LogQueryErrorAsync(
                queryId, user.UserId, ex.Message,
                cancellationToken);

            _logger.LogError(ex, "Query {QueryId}
                failed for user {UserId}", queryId
                , user.UserId);

            return Problem("An error occurred while
                 processing your query. Please try
                 again.");
        }
    }

    [HttpPost("documents")]
    [ProducesResponseType(typeof(
        DocumentIngestionResponse), StatusCodes.
        Status201Created)]
    [ProducesResponseType(typeof(ProblemDetails),
        StatusCodes.Status400BadRequest)]
    [Authorize(Policy = "DocumentIngestion")]
    public async Task<ActionResult<
        DocumentIngestionResponse>>
        IngestDocumentAsync(
        [FromForm] DocumentIngestionRequest request
            ,
        CancellationToken cancellationToken =
            default)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }

        var user = await _userContext.
            GetCurrentUserAsync(User,
            cancellationToken);
        var ingestionId = Guid.NewGuid();

        await _auditLogger.LogIngestionAttemptAsync
            (
            ingestionId, user.UserId, request.File.
                FileName, cancellationToken);

        try
        {
            using var stream = request.File.
                OpenReadStream();
            var response = await _ingestionService.
                IngestDocumentAsync(
                stream, request.File.FileName,
                    request.Classification,
                user, ingestionId,
                    cancellationToken);

            await _auditLogger.
                LogIngestionSuccessAsync(
                ingestionId, user.UserId, response.
                    DocumentId, response.
                    ChunkCount, cancellationToken)
                    ;

            _logger.LogInformation("Document {
                FileName} ingested successfully as
                 {DocumentId} " +
                "with {ChunkCount} chunks for user
                    {UserId}",
                request.File.FileName, response.
                    DocumentId, response.
                    ChunkCount, user.UserId);

            return CreatedAtAction(nameof(
                GetDocumentAsync),
                new { id = response.DocumentId },
                    response);
        }
        catch (Exception ex)
        {
            await _auditLogger.
                LogIngestionErrorAsync(
                ingestionId, user.UserId, ex.
                    Message, cancellationToken);

            _logger.LogError(ex, "Document
                ingestion {IngestionId} failed for
                 user {UserId}",
                ingestionId, user.UserId);

            return Problem("An error occurred while
                 ingesting the document. Please
                try again.");
        }
    }

    [HttpGet("documents/{id:guid}")]
    [ProducesResponseType(typeof(DocumentResponse),
        StatusCodes.Status200OK)]
    [ProducesResponseType(typeof(ProblemDetails),
        StatusCodes.Status404NotFound)]
    public async Task<ActionResult<DocumentResponse
        >> GetDocumentAsync(
        [FromRoute] Guid id,
        CancellationToken cancellationToken =
            default)
    {
        var user = await _userContext.
            GetCurrentUserAsync(User,
            cancellationToken);

        var document = await _queryService.
            GetDocumentAsync(id, user,
            cancellationToken);
        if (document == null)
        {
            return NotFound();
        }

        return Ok(document);
    }
}
```

F. Integration Patterns

The implementation provides comprehensive integration capabilities that enable deployment within existing enterprise infrastructure while maintaining governance requirements.

Identity Provider Integration: The system integrates with enterprise identity providers using standard protocols (SAML, OAuth, OpenID Connect) implemented through .NET 9's authentication middleware. Custom authentication handlers translate external identity information into internal governance contexts.

Document Management System APIs: RESTful APIs provide integration with existing document management systems while maintaining governance boundaries. These APIs implement comprehensive input validation and access control enforcement to ensure that external integrations cannot bypass governance requirements.

Compliance Reporting Interfaces: Specialized APIs support compliance reporting requirements through structured data exports that maintain audit trail integrity. These interfaces provide the data access capabilities necessary for regulatory reporting while ensuring that sensitive information is appropriately protected.

G. Monitoring and Observability

The implementation leverages .NET 9's enhanced observability features to provide comprehensive monitoring capabilities that support both operational management and compliance requirements.

Performance Telemetry: Distributed tracing captures performance characteristics across all system components, enabling operators to identify bottlenecks and optimize performance. Custom metrics track governance-specific performance indicators such as access control decision latency and audit log throughput.

Security Monitoring: Specialized monitoring capabilities track access patterns, identify potential security violations, and provide early warning of unusual system behavior. These capabilities integrate with enterprise security information and event management (SIEM) systems through standard protocols.

Compliance Dashboards: Real-time dashboards provide visibility into compliance posture, including access control coverage, document currency, and audit trail completeness. These dashboards support both operational monitoring and compliance reporting requirements.

H. Error Handling and Resilience

The implementation includes comprehensive error handling and resilience patterns that ensure system availability while maintaining governance requirements even during failure scenarios.

Graceful Degradation: When system components become unavailable, the implementation implements graceful degradation that maintains essential governance functions while potentially reducing performance or feature availability. This approach ensures that access control decisions remain consistent even during partial system failures.

Audit Trail Preservation: Error handling procedures prioritize audit trail preservation, ensuring that even system failures are comprehensively logged. Recovery procedures include audit trail validation to ensure that no governance decisions are lost during failure recovery.

Circuit Breaker Patterns: External dependencies use circuit breaker patterns that prevent cascading failures while maintaining system stability. These patterns include governance-aware fallback procedures that ensure access control decisions can be made even when external systems are unavailable.

I. Dependency Injection Configuration

The implementation leverages .NET 9's dependency injection container to manage service lifetimes and maintain clean separation of concerns:

Listing 6: Service Registration and Configuration

```
public static class ServiceCollectionExtensions
{
    public static IServiceCollection
        AddKnowledgeGovernance(
        this IServiceCollection services,
        IConfiguration configuration)
    {
        // Configuration options
        services.Configure<QdrantOptions>(
            configuration.GetSection("Qdrant"));
        services.Configure<GovernanceOptions>(
            configuration.GetSection("Governance")
            );
        services.Configure<EmbeddingOptions>(
            configuration.GetSection("Embedding"))
            ;

        // Core services
        services.AddScoped<IAccessControlService,
            AccessControlService>();
        services.AddScoped<IVectorService,
            QdrantVectorService>();
        services.AddScoped<IKnowledgeQueryService,
            KnowledgeQueryService>();
        services.AddScoped<
            IDocumentIngestionService,
            DocumentIngestionService>();
        services.AddScoped<IUserContextService,
            UserContextService>();

        // Repository pattern
        services.AddScoped<IDocumentRepository,
            DocumentRepository>();
        services.AddScoped<IChunkRepository,
            ChunkRepository>();
        services.AddScoped<IAuditRepository,
            AuditRepository>();

        // Validation and governance
        services.AddScoped<IContentValidator,
            ContentValidator>();
        services.AddScoped<IPolicyEngine,
            PolicyEngine>();
        services.AddScoped<ILineageTracker,
            LineageTracker>();

        // Logging and monitoring
        services.AddScoped<IAuditLogger,
            AuditLogger>();
        services.AddScoped<IGovernanceMetrics,
            GovernanceMetrics>();

        // Background services
        services.AddHostedService<
            GovernanceAgentService>();
        services.AddHostedService<
            DocumentCurrencyService>();
        services.AddHostedService<
            AuditLogCleanupService>();

        // External integrations
        services.AddSingleton<QdrantClient>(
            provider =>
        {
```

```
            var options = provider.
                GetRequiredService<IOptions<
                QdrantOptions>>().Value;
            return new QdrantClient(options.Host,
                options.Port, options.ApiKey);
        });

        services.AddHttpClient<IEmbeddingService,
            QwenEmbeddingService>(client =>
        {
            var options = configuration.GetSection
                ("Embedding").Get<EmbeddingOptions
                >();
            client.BaseAddress = new Uri(options.
                ServiceUrl);
            client.Timeout = TimeSpan.FromSeconds(
                options.TimeoutSeconds);
        });

        // Entity Framework
        services.AddDbContext<GovernanceDbContext>(
            options =>
        {
            var connectionString = configuration.
                GetConnectionString("
                DefaultConnection");
            options.UseSqlServer(connectionString,
                sql =>
            {
                sql.EnableRetryOnFailure(
                    maxRetryCount: 3);
                sql.CommandTimeout(30);
            });
            options.EnableSensitiveDataLogging(
                false);
            options.EnableServiceProviderCaching();
        });

        // Authentication and authorization
        services.AddAuthentication(
            JwtBearerDefaults.AuthenticationScheme
            )
            .AddJwtBearer(options =>
            {
                var authOptions = configuration.
                    GetSection("Authentication").
                    Get<AuthenticationOptions>();
                options.Authority = authOptions.
                    Authority;
                options.Audience = authOptions.
                    Audience;
                options.RequireHttpsMetadata =
                    authOptions.RequireHttps;
            });

        services.AddAuthorization(options =>
        {
            options.AddPolicy("DocumentIngestion",
                policy =>
                policy.RequireClaim("permissions",
                    "documents:write"));
            options.AddPolicy("AdminOperations",
                policy =>
                policy.RequireClaim("permissions",
                    "admin:all"));
        });

        // Health checks
        services.AddHealthChecks()
            .AddDbContextCheck<GovernanceDbContext
                >()
            .AddCheck<QdrantHealthCheck>("qdrant")
            .AddCheck<EmbeddingServiceHealthCheck
                >("embedding");

        // OpenAPI documentation
        services.AddEndpointsApiExplorer();
        services.AddSwaggerGen(c =>
        {
            c.SwaggerDoc("v1", new OpenApiInfo
            {
                Title = "Knowledge Governance API",
                Version = "v1",
                Description = "Enterprise PDF
                    Knowledge Governance Layer"
            });
            c.AddSecurityDefinition("Bearer", new
                OpenApiSecurityScheme
            {
                Type = SecuritySchemeType.Http,
                Scheme = "bearer",
                BearerFormat = "JWT"
            });
        });

        return services;
    }

    public static async Task<WebApplication>
        ConfigureKnowledgeGovernanceAsync(
        this WebApplication app)
    {
        // Middleware pipeline
        app.UseAuthentication();
        app.UseAuthorization();

        // API documentation
        if (app.Environment.IsDevelopment())
        {
            app.UseSwagger();
            app.UseSwaggerUI();
        }

        // Health checks
        app.UseHealthChecks("/health", new
            HealthCheckOptions
        {
            ResponseWriter = UIResponseWriter.
                WriteHealthCheckUIResponse
        });

        // Controllers
        app.MapControllers();

        // Database migration
        using var scope = app.Services.CreateScope
            ();
        var context = scope.ServiceProvider.
            GetRequiredService<GovernanceDbContext
            >();
        await context.Database.MigrateAsync();

        return app;
    }
}
```

J. Configuration and Deployment

The implementation supports flexible configuration and deployment patterns that enable adaptation to diverse enterprise environments while maintaining consistent governance capabilities.

Configuration Management: Governance policies are managed through configuration files that support version control and deployment automation. The configuration system

includes validation capabilities that ensure policy changes maintain system consistency and compliance requirements.

Container Deployment: The system is designed for container-based deployment using Docker and Kubernetes, with governance metadata stored in persistent volumes that survive container restarts. Container orchestration ensures that governance services maintain high availability while supporting horizontal scaling.

Environment Isolation: Different deployment environments (development, testing, production) maintain separate governance contexts while sharing common policy frameworks. This isolation ensures that testing activities cannot compromise production governance while maintaining consistency across environments.

This implementation design provides the technical foundation necessary for deploying the confidential PDF knowledge governance layer in enterprise environments. The design ensures that conceptual governance guarantees are realized through practical implementation choices that prioritize reliability, maintainability, and compliance while delivering the performance characteristics necessary for enterprise adoption.

K. Alternative Implementation Approaches

Software engineers have several options for implementing access control in vector search systems. This section compares the payload-based approach with common alternatives to help development teams make informed architectural decisions.

1) Application-Layer Filtering: **Implementation**: Perform vector search without access control, then filter results in application code based on user permissions.

Advantages:

- Simple to implement initially
- Works with any vector database
- Familiar pattern for most developers
- Easy to debug and test

Disadvantages:

- Security vulnerability: unauthorized data may be returned to application layer
- Performance issues: must retrieve more results than needed, then filter
- Inconsistent result counts: user sees fewer results than requested
- Complex caching: cannot cache filtered results safely

When to Use: Proof-of-concept systems or environments where all users have similar access levels.

2) Separate Authorization Service: **Implementation**: Query external authorization service for each document/chunk before including in vector search results.

Advantages:

- Centralized authorization logic
- Consistent with enterprise authorization patterns
- Can leverage existing identity management systems
- Clear separation of concerns

Disadvantages:

- Network latency for each authorization check
- Complex error handling for authorization service failures
- Difficult to optimize for performance
- Cannot pre-filter vector search results

When to Use: Systems with complex, dynamic authorization rules that change frequently.

3) Database Views or Query Rewriting: **Implementation**: Use database views or query rewriting to automatically add access control filters to vector queries.

Advantages:

- Transparent to application code
- Leverages database security features
- Consistent with traditional database security patterns
- Can be enforced at database level

Disadvantages:

- Limited support in vector databases
- Complex to implement with semantic similarity queries
- Difficult to debug and troubleshoot
- May not work with all vector database features

When to Use: Organizations with strong database administration teams and simple access control requirements.

4) Payload-Based Filtering (Our Approach): **Implementation**: Embed access control metadata in vector database payloads and use native filtering capabilities.

Advantages:

- Security: access control enforced at database level
- Performance: filtering occurs before similarity computation
- Consistency: guaranteed result counts match user expectations
- Auditability: clear trail of access control decisions

Disadvantages:

- Storage overhead for metadata
- Requires vector database with payload filtering support
- More complex initial setup
- Metadata synchronization challenges during updates

When to Use: Production enterprise systems requiring strong security guarantees and consistent performance.

5) Hybrid Approaches: Many production systems combine multiple approaches:

Payload + Application Layer: Use payload filtering for primary access control, with application-layer checks for complex business rules.

Payload + Caching: Use payload filtering with intelligent caching of access control decisions to improve performance.

Tiered Access Control: Use document-level permissions for coarse filtering, then chunk-level payload filtering for fine-grained control.

6) Decision Framework: Choose your approach based on:

- **Security Requirements**: How critical is it to prevent unauthorized access?
- **Performance Needs**: What are your latency and throughput requirements?
- **Access Pattern Complexity**: How often do permissions change?

- **Team Expertise**: What are your team's strengths in database vs. application development?
- **Existing Infrastructure**: What authorization systems are already in place?

L. Common Implementation Challenges and Solutions

Development teams implementing access-controlled vector search systems encounter several recurring challenges. This section provides practical solutions based on common enterprise deployment patterns.

1) Metadata Synchronization: **Challenge**: Keeping access control metadata synchronized with source document permissions when permissions change frequently.

Common Failure Modes:

- User permissions change in Active Directory but vector database metadata is not updated
- Document access levels change but existing chunks retain old permissions
- Bulk permission updates cause temporary inconsistencies

Solution Pattern - Event-Driven Updates:

Listing 7: Permission Synchronization Service

```
public class PermissionSyncService :
    BackgroundService
{
    private readonly IServiceBus _serviceBus;
    private readonly IVectorService _vectorService;

    protected override async Task ExecuteAsync(
        CancellationToken stoppingToken)
    {
        await _serviceBus.SubscribeAsync<
            PermissionChangedEvent>(
            async (evt) => await
                HandlePermissionChange(evt));
    }

    private async Task HandlePermissionChange(
        PermissionChangedEvent evt)
    {
        var chunks = await _vectorService
            .GetChunksByDocumentId(evt.DocumentId);
        foreach (var chunk in chunks)
        {
            chunk.Payload["access_roles"] = evt.
                NewAccessRoles;
            await _vectorService.UpdateChunk(chunk)
                ;
        }
    }
}
```

2) Performance Optimization: **Challenge**: Access control filtering can significantly impact query performance, especially with complex role hierarchies.

Common Performance Issues:

- Large role lists in payload filters cause slow queries
- Complex role inheritance rules require expensive computations
- High-cardinality access control fields reduce index effectiveness

Solution Pattern - Role Flattening:

Listing 8: Role Flattening for Performance Optimization

```
public class RoleFlattener
{
    public List<string> FlattenUserRoles(
        ClaimsPrincipal user)
    {
        var roles = new HashSet<string>();

        // Direct roles
        roles.UnionWith(user.FindAll(ClaimTypes.
            Role)
            .Select(c => c.Value));

        // Inherited roles (pre-computed)
        var inheritedRoles = _roleHierarchy
            .GetInheritedRoles(user.Identity.Name);
        roles.UnionWith(inheritedRoles);

        // Group memberships (flattened)
        var groupRoles = _groupService
            .GetEffectiveRoles(user.Identity.Name);
        roles.UnionWith(groupRoles);

        return roles.ToList();
    }
}
```

3) Debugging and Troubleshooting: **Challenge**: When users cannot find expected documents, it's difficult to determine whether the issue is with embeddings, access control, or document processing.

Solution Pattern - Comprehensive Logging:

Listing 9: Comprehensive Search Logging

```
public async Task<SearchResult> SearchAsync(
    string query, ClaimsPrincipal user)
{
    var searchId = Guid.NewGuid();
    var userRoles = _roleFlattener.FlattenUserRoles
        (user);

    _logger.LogInformation(
        "Search {SearchId}: User {User} with roles
            {Roles}",
        searchId, user.Identity.Name,
        string.Join(",", userRoles));

    var filter = CreateAccessFilter(userRoles);
    var results = await _vectorService.SearchAsync(
        query, filter);

    _logger.LogInformation(
        "Search {SearchId}: Found {Count} results",
        searchId, results.Count);

    if (_logger.IsEnabled(LogLevel.Debug))
    {
        var unfiltered = await _vectorService
            .SearchAsync(query, null);
        var excluded = unfiltered.Count - results.
            Count;
        _logger.LogDebug("Search {SearchId}: {
            Excluded} excluded",
            searchId, excluded);
    }
    return results;
}
```

4) Testing Access Control Logic: **Challenge**: Testing access control requires creating realistic user scenarios and verifying that unauthorized content is never returned.

Solution Pattern - Role-Based Test Fixtures:

Listing 10: Role-Based Access Control Testing

```
[Test]
public async Task SearchShouldRespectRoleBasedAccess()
{
    // Arrange
    var hrDocument = await CreateTestDocument(
        "HR Policy", roles: ["HR", "Management"]);
    var publicDocument = await CreateTestDocument(
        "Public Announcement", roles: ["Everyone"]);

    var hrUser = CreateTestUser(roles: ["HR"]);
    var regularUser = CreateTestUser(roles: ["Employee"]);

    // Act
    var hrResults = await _searchService
        .SearchAsync("policy", hrUser);
    var regularResults = await _searchService
        .SearchAsync("policy", regularUser);

    // Assert
    Assert.That(hrResults.Any(r =>
        r.DocumentId == hrDocument.Id), Is.True);
    Assert.That(regularResults.Any(r =>
        r.DocumentId == hrDocument.Id), Is.False);
    Assert.That(regularResults.Any(r =>
        r.DocumentId == publicDocument.Id), Is.True);
}
```

5) Monitoring and Alerting: **Challenge**: Production systems need monitoring to detect access control failures, performance degradation, and potential security issues.

Solution Pattern - Access Control Metrics:

Listing 11: Access Control Monitoring Metrics

```
public class AccessControlMetrics
{
    private readonly IMetrics _metrics;

    public void RecordSearchFiltering(int totalResults,
        int filteredResults, TimeSpan filterTime)
    {
        _metrics.Counter("search.results.total")
            .Increment(totalResults);
        _metrics.Counter("search.results.filtered")
            .Increment(filteredResults);
        _metrics.Timer("search.filter.duration")
            .Record(filterTime);

        var exclusionRate = (double)(totalResults -
            filteredResults)
            / totalResults;
        _metrics.Gauge("search.exclusion_rate")
            .Set(exclusionRate);

        // Alert if exclusion rate is unusually high
        if (exclusionRate > 0.8)
        {
            _metrics.Counter("search.high_exclusion_rate")
                .Increment();
        }
    }
}
```

6) Deployment and Migration: **Challenge**: Migrating existing vector databases to include access control metadata without service disruption.

Solution Pattern - Blue-Green Migration:

Listing 12: Blue-Green Access Control Migration

```
public class AccessControlMigration
{
    public async Task MigrateToAccessControlled()
    {
        // Phase 1: Add metadata to new documents
        _documentProcessor.UseAccessControlMetadata
            = true;

        // Phase 2: Backfill existing documents
        var existingChunks = await _vectorService.
            GetAllChunks();
        foreach (var chunk in existingChunks)
        {
            if (!chunk.Payload.ContainsKey("
                access_roles"))
            {
                var document = await
                    _documentService
                    .GetDocument(chunk.DocumentId);
                chunk.Payload["access_roles"] =
                    document.AccessRoles;
                await _vectorService.UpdateChunk(
                    chunk);
            }
        }

        // Phase 3: Enable access control filtering
        _searchService.UseAccessControlFiltering =
            true;
    }
}
```

These implementation patterns address the most common challenges development teams face when deploying access-controlled vector search systems in production environments. Each pattern includes error handling, logging, and monitoring considerations essential for enterprise deployment.

VII. Evaluation Framework

This section outlines how development teams can evaluate the proposed architectural approach in their specific enterprise environments. Rather than presenting experimental results, we provide a conceptual framework for validation that teams can adapt to their requirements.

A. Validation Approach for Development Teams

Recommended Test Environment: Development teams should create a representative test corpus reflecting their enterprise document structure. A typical evaluation might

include documents across multiple access levels (e.g., Public, Internal, Confidential, Restricted) with realistic role hierarchies matching organizational structure.

Implementation Comparison: Teams can compare the payload-based approach against alternatives they're considering:

- **Application-Layer Filtering**: Implementing access control in application code after vector search
- **External Authorization Service**: Using separate authorization calls for each search result
- **Document-Level Permissions**: Coarse-grained access control at document level only
- **Payload-Based Filtering**: The approach described in this paper

B. Key Properties to Validate

Development teams implementing this approach should validate the following properties in their specific enterprise context:

Access Control Correctness: Verify that the access control implementation correctly enforces organizational policies. Teams should test representative user roles against documents with different access levels to ensure unauthorized content is never returned.

Consistency During Updates: Test concurrent document updates to ensure metadata synchronization works correctly. This includes scenarios where user permissions change while documents are being processed or updated.

Performance Acceptability: Measure query performance with access control filtering enabled compared to unfiltered search. Key metrics include response time, throughput under load, and storage overhead for metadata.

Integration Compatibility: Validate integration with existing enterprise systems including Active Directory, document management systems, and compliance reporting tools. Ensure the approach works with current authentication and authorization infrastructure.

C. Implementation Considerations and Limitations

Development teams should be aware of several considerations when implementing this approach:

Domain-Specific Complexity: Each organization has unique access control requirements that may not be captured by the general framework presented here. Teams should carefully analyze their specific organizational policies and compliance requirements.

Scale Considerations: The approach is designed for typical enterprise document volumes, but organizations with millions of documents or complex distributed deployments may need additional architectural considerations not covered in this framework.

Semantic Search Quality: Access control filtering may impact search relevance for authorized users if it significantly reduces the candidate set. Teams should monitor search quality metrics after implementation.

User Experience Impact: The administrative overhead of maintaining access control metadata should be evaluated against the security benefits. Consider the impact on document management workflows.

Vendor-Specific Features: This framework focuses on Qdrant, but teams using other vector databases should evaluate whether similar payload filtering capabilities are available and how they might differ.

Teams implementing this approach should plan for thorough testing in their specific environment and consider starting with a pilot deployment to validate the approach before full-scale implementation.

VIII. Related Work

This work builds on established research in access control, retrieval-augmented generation, and enterprise document management systems. We organize related work into three categories that highlight different approaches to the core challenge.

A. Vector Database Access Control

Vector-First Systems: LangChain [11] and similar frameworks provide semantic search capabilities but lack integrated access control. These systems treat permissions as external concerns, leading to implementation complexity and potential security gaps. Modern vector databases excel at billion-scale similarity search [1] but were not designed with enterprise security requirements in mind.

Traditional Enterprise Search: Enterprise document management systems like SharePoint and Documentum offer comprehensive governance features including role-based access control, version management, and compliance reporting, but cannot support the natural language interactions that vector databases enable. The semantic gap between traditional keyword-based search and natural language queries creates user experience limitations that our framework addresses.

Hybrid Approaches: Recent systems attempt to bridge vector databases with traditional access control by implementing application-layer filtering. However, these approaches suffer from performance overhead and complexity in maintaining consistency between permission systems and vector embeddings. The fundamental challenge lies in reconciling the approximate nature of vector similarity search [2] with the precise requirements of access control systems.

B. Access Control Models

The foundational work on role-based access control (RBAC) by Sandhu et al. [8] provides the theoretical foundation for hierarchical permission models. The ARBAC97 administrative model [12] addresses role administration in large organizations, concepts that our framework extends to vector database environments. Early work by Ferraiolo and Kuhn [13] established the basic principles that continue to guide enterprise access control systems.

The NIST guide to attribute-based access control (ABAC) [14] provides theoretical foundations for the contextual access decisions our framework implements. However,

existing ABAC research focuses on structured data rather than the semantic similarity relationships that define vector database operations. Earlier work by Yuan and Tong [15] explored ABAC for web services, establishing patterns that inform modern enterprise access control architectures.

Temporal access control models [9] introduce time-aware access control decisions that consider when information was created, modified, and accessed. This work provides important foundations for the temporal governance capabilities our framework implements.

C. Explainable AI and Audit Systems

Explainable AI research has developed techniques for providing transparency into AI decision-making processes. The LIME [16] and SHAP [17] frameworks enable understanding of individual AI predictions. Our framework builds on this work by providing audit trails that capture not only what information was accessed but why specific documents were considered relevant to user queries.

Gap Analysis: While existing research addresses individual components of the access control challenge, no prior work provides a comprehensive framework that integrates vector database operations with enterprise governance requirements. Our contribution lies in demonstrating how role-based access control can be efficiently implemented at the vector database level while maintaining the performance characteristics that make semantic search valuable for enterprise applications.

D. Retrieval-Augmented Generation Systems

We systematically analyze existing approaches to enterprise document management with semantic capabilities, identifying specific technical gaps that our work addresses. The foundational RAG architecture introduced by Lewis et al. [18] established the pattern of combining parametric and non-parametric knowledge, but did not address enterprise access control requirements. Dense passage retrieval [19] improved retrieval effectiveness but assumes uniform access to all passages.

Cloud Document AI: SharePoint Syntex, Google Cloud Document AI, and AWS Textract with IAM integration provide both access control and semantic processing. However, these solutions operate at the document level rather than the chunk level, limiting their applicability to scenarios requiring fine-grained access control within documents. Additionally, they are cloud-specific and cannot be deployed in air-gapped enterprise environments.

Our approach differs by implementing access control directly in the vector database payload, enabling atomic operations where access decisions and semantic ranking occur simultaneously within the same database transaction. This leverages advances in transformer-based embeddings [20], [21] that enable semantic search while maintaining compatibility with traditional database filtering operations.

E. Enterprise Document Management

Traditional enterprise document management systems provide robust access control and audit capabilities but lack the semantic understanding necessary for AI-powered interactions. Systems like SharePoint and Documentum offer comprehensive governance features including role-based access control, version management, and compliance reporting, but cannot support the natural language interactions that make AI valuable for knowledge work.

Modern enterprise content management platforms face the challenge of integrating semantic search capabilities while maintaining existing governance structures. This creates a fundamental tension between the flexibility required for AI-powered interactions and the control required for enterprise compliance.

F. Access Control in Information Systems

Fine-grained access control has been extensively studied in database and information systems research. The concept of attribute-based access control (ABAC) [14] provides theoretical foundations for the contextual access decisions our framework implements. However, existing ABAC research focuses on structured data rather than the semantic chunks that characterize AI-powered document systems.

Traditional role-based access control models work well for structured systems but face challenges when applied to vector databases where similarity relationships rather than explicit relationships determine data access patterns. Our framework addresses this gap by implementing access control at the vector payload level.

G. AI Governance and Explainable Systems

The field of AI governance has emerged as organizations grapple with the challenges of deploying AI systems in regulated environments. Enterprise AI governance requires balancing innovation with compliance, transparency, and accountability requirements that traditional software governance frameworks do not address. The rapid advancement of large language models [22]–[24] has accelerated enterprise adoption while intensifying governance challenges.

Explainable AI research has developed techniques for providing transparency into AI decision-making processes. The LIME [16] and SHAP [17] frameworks enable understanding of individual AI predictions. Our framework builds on this work by providing audit trails that capture not just retrieval decisions but also the governance processes that determine what information is available for retrieval.

H. Vector Database Security

As vector databases become increasingly important for AI applications, their unique security requirements have become apparent. Unlike traditional databases where access control operates on discrete records, vector databases require access control mechanisms that respect similarity relationships while maintaining security boundaries.

Current vector database implementations like Qdrant and Pinecone provide basic payload filtering capabilities, but these are typically designed for functional rather than security purposes. Our framework demonstrates how these capabilities

can be extended to provide enterprise-grade access control. This represents an evolution from traditional term-weighting approaches [25] to modern semantic embeddings while maintaining the filtering capabilities essential for enterprise deployment.

I. Compliance and Audit Systems

Enterprise compliance systems have developed sophisticated approaches to audit trail generation and regulatory reporting. Traditional compliance frameworks focus on structured transactions and explicit user actions, but AI systems require audit capabilities that can capture the semantic relationships and inference processes that influence system behavior.

Modern compliance requirements increasingly demand not just logs of what happened, but explanations of why specific decisions were made. This creates new challenges for AI systems where the "decision" process involves semantic similarity calculations rather than explicit rule evaluation.

J. Differentiation and Contributions

While each of these research areas provides important foundations for our work, no existing system addresses the comprehensive governance requirements that enterprise AI deployment demands. Traditional document management systems lack semantic understanding, RAG systems lack governance capabilities, and AI governance frameworks lack the operational specificity necessary for document-centric applications.

Our framework's primary contribution lies in synthesizing concepts from these diverse research areas into a coherent conceptual foundation that addresses the practical challenges of enterprise AI deployment. By treating governance and AI capabilities as co-equal design requirements, we provide a foundation for systems that can deliver the user experience benefits of AI while maintaining the trust and compliance requirements necessary for enterprise adoption.

The conceptual framework we present enables organizations to move beyond the current choice between ungoverned AI systems that cannot be deployed in regulated environments and traditional document management systems that cannot support AI-powered interactions. Instead, we provide a foundation for systems that deliver both AI capabilities and enterprise governance requirements through integrated design rather than retrofitted additions.

IX. Conclusion

Software development teams building enterprise RAG systems face a critical implementation challenge: integrating role-based access control with vector databases while maintaining both security and performance. This paper provides a practical architectural framework that development teams can immediately apply to production systems.

Our engineering contributions provide actionable guidance for software teams:

Implementation Architecture: Complete .NET code examples demonstrating payload-based access control integration with Qdrant, including dependency injection patterns, background services, and API controllers ready for production deployment.

Alternative Approach Analysis: Systematic comparison of different access control implementation strategies with clear guidance on when to use each approach based on security requirements, performance needs, and team expertise.

Common Challenge Solutions: Practical solutions to recurring implementation problems including metadata synchronization, performance optimization, debugging strategies, testing patterns, and deployment migration approaches.

Decision Framework: Clear criteria for choosing between implementation approaches based on enterprise requirements, existing infrastructure, and development team capabilities.

A. Limitations

Our approach has significant limitations that constrain its applicability:

Limited Novelty: The core technique—using database payload filtering for access control—is not novel. Our contribution is applying this existing capability to semantic search rather than developing new algorithms.

Conceptual Scope: Framework addresses basic role-based access control scenarios. Real enterprise environments involve complex, dynamic access requirements including attribute-based access control, temporal restrictions, and cross-organizational policies not addressed in our model.

Semantic Quality Unknown: We have not measured whether access restrictions degrade search quality for authorized users, which is critical for practical deployment.

Scalability Analysis: Conceptual framework does not address distributed system challenges including consistency across multiple nodes, network partitions, and geographic distribution.

Implementation Gap: While we provide .NET code examples, full enterprise deployment requires integration with existing identity management, compliance systems, and operational infrastructure not addressed in our work.

B. Future Work

Critical next steps include:

Empirical Validation: Implementation and testing with real enterprise document corpora to validate theoretical properties and measure performance characteristics.

Semantic Quality Analysis: Investigating whether access restrictions degrade search relevance for authorized users and developing mitigation strategies.

Extended Access Models: Incorporating attribute-based access control, temporal restrictions, and dynamic policy updates into the mathematical framework.

Distributed Systems: Extending the consistency model to multi-node deployments with network partitions and geographic distribution.

Enterprise Integration: Developing integration patterns with existing identity management systems, compliance frameworks [26], and operational infrastructure that address

regulatory requirements such as GDPR [27] and Sarbanes-Oxley [28].

This architectural framework addresses a genuine gap in enterprise RAG system development. While vector database tutorials focus on basic similarity search, production systems require access control integration that most development teams must solve independently. This paper consolidates proven patterns and provides implementation guidance that development teams can adapt to their specific enterprise requirements.

The value lies not in algorithmic innovation, but in providing software engineers with battle-tested architectural patterns, comprehensive code examples, and practical decision-making frameworks for a common enterprise development challenge.

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," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[3] P. Samarati and S. De Capitani di Vimercati, "Access control: principle and practice," *IEEE communications magazine*, vol. 39, no. 9, pp. 148–153, 2001.

[4] M. Fowler, *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.

[5] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[6] E. F. Codd, "A relational model of data for large shared data banks," *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, 1970.

[7] S. Newman, *Building microservices: designing fine-grained systems*. O'Reilly Media, 2015.

[8] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[9] E. Bertino, P. A. Bonatti, and E. Ferrari, "A temporal role-based access control model," *ACM Transactions on Information and System Security*, vol. 4, no. 3, pp. 191–233, 2001.

[10] Microsoft, "C# language specification," https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/, 2023.

[11] LangChain, "Langchain," https://github.com/langchain-ai/langchain, 2023.

[12] R. S. Sandhu, V. Bhamidipati, and Q. Munawer, "The arbac97 model for role-based administration of roles," *ACM Transactions on Information and System Security*, vol. 2, no. 1, pp. 105–135, 1999.

[13] D. F. Ferraiolo and D. R. Kuhn, "Role-based access control," in *Proceedings of the 15th NIST-NCSC National Computer Security Conference*, 1992, pp. 554–563.

[14] V. C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone, "Guide to attribute based access control (abac) definition and considerations," National Institute of Standards and Technology, Tech. Rep. NIST Special Publication 800-162, 2014.

[15] E. Yuan and J. Tong, "Attributed based access control (abac) for web services," in *IEEE International Conference on Web Services (ICWS'05)*. IEEE, 2005, pp. 561–569.

[16] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should i trust you?: Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 1135–1144.

[17] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

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

[19] V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih, "Dense passage retrieval for open-domain question answering," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020, pp. 6769–6781.

[20] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[21] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, 2019.

[22] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Saxena, A. Sharma *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[24] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[25] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information processing & management*, vol. 24, no. 5, pp. 513–523, 1988.

[26] National Institute of Standards and Technology, "Framework for improving critical infrastructure cybersecurity," National Institute of Standards and Technology, Tech. Rep. NIST Cybersecurity Framework Version 1.1, 2018.

[27] European Parliament, "General data protection regulation (gdpr)," Regulation (EU) 2016/679, 2016.

[28] US Congress, "Sarbanes-oxley act of 2002," *Public Law*, vol. 107, p. 204, 2002.

Access-Controlled Semantic Search.pdf

ORIGINALITY REPORT

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

PRIMARY SOURCES

1	arxiv.org Internet Source	2%
2	www.medrxiv.org Internet Source	1%
3	antonio-cau.co.uk Internet Source	1%
4	www.biorxiv.org Internet Source	<1%
5	research-portal.uu.nl Internet Source	<1%
6	Submitted to University of Ulster Student Paper	<1%
7	www.cs.uic.edu Internet Source	<1%
8	ela.kpi.ua Internet Source	<1%
9	"ITNG 2024: 21st International Conference on Information Technology-New Generations", Springer Science and Business Media LLC, 2024 Publication	<1%
10	www.iaas.uni-stuttgart.de Internet Source	<1%
11	export.arxiv.org Internet Source	<1%
12	core.ac.uk	

	Internet Source	<1%
13	epub.uni-regensburg.de Internet Source	<1%
14	dev.to Internet Source	<1%
15	www.dcs.bbk.ac.uk Internet Source	<1%
16	dspace.univ-msila.dz Internet Source	<1%
17	Submitted to ICTS Student Paper	<1%
18	Submitted to Polytechnic of Zagreb Student Paper	<1%
19	feed.nuget.org Internet Source	<1%
20	profsandhu.com Internet Source	<1%
21	s3.amazonaws.com Internet Source	<1%
22	scholar.harvard.edu Internet Source	<1%
23	www.yumpu.com Internet Source	<1%
24	Leihong Wu, Joshua Xu, Shraddha Thakkar, Magnus Gray, Yanyan Qu, Dongying Li, Weida Tong. "A framework enabling LLMs into regulatory environment for transparency and trustworthiness and its application to drug labeling document", Regulatory Toxicology and Pharmacology, 2024 Publication	<1%

25	rehansaeed.com Internet Source	<1%
26	teaching.ust.hk Internet Source	<1%
27	Bhavani Thuraisngham, Murat Kantarcioglu, Latifur Khan. "Secure Data Science - Integrating Cyber Security and Data Science", CRC Press, 2022 Publication	<1%
28	Isaac Agudo. "Enabling Attribute Delegation in Ubiquitous Environments", Mobile Networks and Applications, 07/04/2008 Publication	<1%
29	Salwa Belaqziz, Salma El Hajjami, Hicham Amellal, Redouan Lahmyed, Lahcen Koutti, Boulouz Abdellah, Inam Ullah Khan. "Smart Applications of Artificial Intelligence and Big Data", CRC Press, 2025 Publication	<1%
30	harvest.usask.ca Internet Source	<1%
31	leanpub.com Internet Source	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		