Legal Document Fraud Detection Semantic Analysis Pipeline Using .NET Core 9, OCR, Vector Search and Open Qwen LLMs.pdf

by Turnitin Student

Submission date: 02-Jun-2025 09:17PM (UTC-0400)

Submission ID: 2690958492

File name: Legal_Document_Fraud_Detection_Semantic_Analysis_Pipeline_Using_.NET_Core_9_OCR_Vector_Search_and_Open_Qwen_LLMs.pdf (425.57K)

Word count: 10157

Character count: 64893

Legal Document Fraud Detection: Semantic Analysis Pipeline Using .NET Core 9, OCR, Vector Search, and Open Qwen LLMs

Mateus Yonathan
Software Developer & Independent Researcher

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

***Abstract*—As legal document fraud becomes more sophisticated, perpetrators increasingly rely on semantic manipulation, where clauses are subtly reworded within scanned or digital PDFs to create deceptive yet legally plausible content. This theoretical study presents an enterprise-grade document fraud detection pipeline built in .NET Core 9, combining open-source PDF parsing, high-accuracy OCR, vector similarity search, and transformer-based language models. The architecture includes modular ASP.NET Core microservices, resilient background workers using Wolverine and Microsoft.Extensions.Http.Resilience, and a Qdrant vector database for clause-level semantic retrieval. For text-based PDFs, PdfPig or PdfSharpCore is used to extract structured content. For scanned PDFs, Tesseract OCR (via .NET bindings) converts image pages into searchable text. Extracted clauses are segmented and embedded using the open Qwen model family, such as 'qwen-embedding' or 'qwen2.5-7b-instruct', which provide strong semantic representation without relying on closed APIs. These embeddings are compared against a trusted corpus to detect subtle alterations. A local LLM component then analyzes and explains suspicious clause deviations in natural language. A hypothetical case study involving reworded indemnity clauses in apartment sales contracts demonstrates the system's potential. Although this pipeline remains theoretical, it outlines a scalable, interpretable, and fully open-source foundation for document fraud detection in modern .NET enterprise environments.**

I. Introduction

Legal documents serve as the backbone of today's business transactions and agreements - no question about it. Think about everything from real estate deals to service contracts; these papers create binding commitments that shape business relationships and spell out who's responsible for what. But here's the problem: as we've gone increasingly digital with these materials, we're seeing a troubling trend emerge - sophisticated fraud through clever semantic manipulation [1].

Gone are the days of crude forgeries with obviously altered signatures or dates. No, modern document fraud is far sneakier. Bad actors make strategic, often tiny rewording changes to key clauses that completely flip the meaning and obligations while keeping the document looking legitimate on the surface [2]. And let's be honest - these subtle changes, buried in long documents filled with dense legalese, often slip right past manual reviews.

A. Core Mathematical Model

Our approach models document fraud detection as a vector similarity problem in high-dimensional semantic space. For a pair of clauses (c_1, c_2), we compute:

$$F(c_1, c_2) = \begin{cases} 1 & \text{if } sim(E(c_1), E(c_2)) < \tau \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Where $E(c)$ represents the embedding function mapping clause text to a vector, sim is a similarity function (typically cosine similarity), and τ is a learned threshold that indicates potential fraud when similarity falls below it. This mathematical foundation enables our system to detect subtle semantic manipulations while ignoring stylistic variations.

B. Problem Context

Here's a typical scenario to consider: Imagine a real estate transaction with a massive 50-page purchase agreement. Somewhere on page 37, there's a single indemnity clause that's been subtly tweaked to shift liability from the seller to the buyer [3]. Maybe just three or four carefully chosen words were changed, but the financial and legal consequences? Potentially huge. Traditional verification methods that just check signatures and document integrity at the binary level would completely miss this kind of semantic manipulation.

This paper lays out a theoretical framework for an enterprise-grade document fraud detection system built specifically to tackle these sophisticated threats. We've leveraged modern tech from the .NET Core 9 ecosystem [4] and combined them into a pipeline that can process, analyze, and flag suspicious semantic alterations in legal documents.

Our proposed system brings together several key technologies:

- Open-source PDF parsing libraries - think PdfPig [5] and PdfSharpCore - for handling digital documents
- Tesseract OCR [6] with .NET bindings to tackle scanned documents
- Semantic embedding generation using those impressive open Qwen models [7]
- Vector similarity search powered by Qdrant [8] for quick, efficient clause comparison

- Local LLM reasoning to analyze and explain potential fraudulent modifications [9]

What makes our approach different? It's all about semantic understanding rather than simple text matching [10]. We break documents into logical clause chunks, embed their meaning in vector space, and compare them against trusted reference materials. The result? We can spot subtle alterations that might change contractual meaning while looking perfectly fine on the surface.

Look, we're not claiming this is a deployed system - it's theoretical. But it does offer a solid blueprint for organizations wanting better document verification. The best part? The architecture we describe can be implemented using current open-source technologies, so enterprises don't need to depend on proprietary, closed-source APIs.

In the next few sections, we'll dive into the background and related work, detail the system architecture, walk through the document processing pipeline, explain the semantic embedding process, outline how the LLM-based reasoning component works, present a case study, and discuss limitations and future research directions.

II. Background and Related Work

Document fraud detection has evolved considerably over the decades, responding to increasingly sophisticated methods of forgery and manipulation. To understand the significance of our proposed approach, it's worth examining this evolution and placing our work in context.

A. Traditional Document Verification

Early approaches to document verification focused primarily on physical authentication elements. These include watermarks, specialized papers, holograms, and signature verification. As documents transitioned to digital formats, the focus shifted toward cryptographic validation methods. Digital signatures, hash-based integrity checking, and Public Key Infrastructure (PKI) became standard tools for verifying document authenticity.

While these methods excel at detecting unauthorized document alterations at a binary level, they fundamentally assume an all-or-nothing approach to document integrity. A document is either completely authentic (bit-for-bit identical to the original) or it's been tampered with. This binary distinction fails to address the more nuanced threat of semantic manipulation, where the document structure remains intact but subtle rewording changes the meaning significantly.

B. Computer Vision and OCR in Document Analysis

For scanned or image-based documents, research in computer vision has yielded significant advances. Early Optical Character Recognition (OCR) systems focused primarily on character-level accuracy, gradually improving to word and sentence-level understanding. Modern OCR systems like Tesseract have achieved impressive accuracy, particularly when fine-tuned for specific document types or languages.

These advancements have enabled the conversion of image-based documents into machine-readable text, opening the door for more sophisticated semantic analysis. However, OCR alone cannot determine whether the content has been altered from an original version unless it has access to that reference material.

C. Natural Language Processing for Legal Documents

The legal domain presents unique challenges for Natural Language Processing (NLP). Legal language is precise, often archaic, and relies heavily on domain-specific terminology. Early NLP approaches to legal document analysis focused on keyword extraction, entity recognition, and basic classification tasks.

Recent research has applied more advanced techniques like dependency parsing and semantic role labeling to understand the structure of legal arguments and obligations. These methods help identify the relationships between entities and actions described in legal text, but they generally lack the ability to compare documents for subtle semantic alterations.

D. Vector Embeddings and Semantic Search

The advent of neural network-based language models has revolutionized text representation through vector embeddings. Starting with models like Word2Vec and GloVe, and advancing to transformer-based approaches like BERT and more recently GPT and Qwen, these models encode semantic meaning into dense vector representations.

Vector embeddings have enabled semantic search capabilities where documents can be matched based on meaning rather than exact keyword matches. Systems like Elasticsearch with vector search capabilities and dedicated vector databases like Qdrant, Weaviate, and Pinecone have made these techniques accessible for production applications.

In the legal domain, vector embeddings have been applied to case law retrieval, contract analysis, and compliance monitoring. However, their application to document fraud detection, particularly for identifying subtle clause alterations, remains relatively unexplored.

E. Language Models for Legal Understanding

Large Language Models (LLMs) have demonstrated remarkable capabilities in understanding and generating legal text. Models like GPT and Qwen can draft contracts, summarize legal documents, and answer complex legal questions with reasonable accuracy. More importantly for our purposes, they can compare text passages and explain semantic differences in natural language.

Open-source models such as the Qwen family have made these capabilities accessible without reliance on closed APIs, enabling on-premises deployment for sensitive legal applications. This democratization of advanced language models has created new possibilities for automated legal document analysis within secure enterprise environments.

F. Existing Commercial Solutions

Several commercial products address aspects of document fraud detection, though most focus on identity documents rather than complex legal contracts. These systems typically combine computer vision, OCR, and pattern matching to detect known forgery techniques. Some advanced solutions incorporate machine learning for anomaly detection, but few explicitly target the semantic manipulation of contract clauses.

Enterprise document management systems often include version control and comparison features, but these typically highlight all textual differences without distinguishing between significant semantic alterations and inconsequential formatting changes.

G. Research Gap

Our review of existing approaches reveals a significant gap: the lack of specialized systems for detecting semantic manipulation in legal documents that:

- Function at the clause level rather than treating documents as atomic units
- Distinguish between semantically significant and trivial textual changes
- Provide natural language explanations of detected alterations
- Operate in a fully open-source stack without reliance on closed APIs
- Integrate into enterprise .NET environments with appropriate governance controls

The approach described in this paper aims to address these limitations by combining modern OCR, vector embeddings, similarity search, and language model reasoning into a cohesive, enterprise-ready pipeline specifically designed for detecting subtle semantic fraud in legal documents.

III. System Architecture

Our proposed legal document fraud detection system is designed as a modern, scalable microservices architecture built on the .NET Core 9 ecosystem. This section outlines the overall system design, component interactions, and technical considerations that inform our theoretical implementation.

A. Architectural Overview

The system follows a modular design with clearly separated concerns, allowing for flexibility, scalability, and maintainability. Figure 1 illustrates the high-level architecture.

B. Mathematical System Model

We model our system as a directed graph $G = (V, E)$ where V represents microservices and E represents communication paths [11]. The workflow function W maps input document d to analysis result r:

$$W(d) = m_n \circ m_{n-1} \circ ... \circ m_1(d) \quad (2)$$

$$R_{sys}(t) = f(R_1(t), R_2(t), ..., R_n(t)) \quad (3)$$

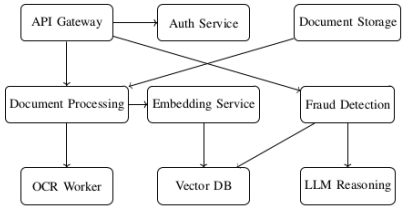

Fig. 1: High-level architecture of the document fraud detection system

C. Core Components

1) API Gateway: The system exposes its functionality through an ASP.NET Core API Gateway that handles request routing, load balancing, and basic request validation [13]. This gateway implements rate limiting, authentication validation, and request logging.

Listing 1: API Gateway Endpoint Implementation

```
public static class DocumentEndpoints
{
    public static void MapDocumentEndpoints(this
        WebApplication app)
    {
        app.MapPost("/api/documents", async (
            HttpRequest request,
            IDocumentService documentService,
            ClaimsPrincipal user) =>
        {
            // Authorization check
            if (!user.HasPermission(Permissions.
                SubmitDocuments))
                return Results.Forbid();

            try {
                // Parse document from request
                var document = await request.
                    ReadFormFileAsync("document");

                // Validate document
                if (!document.IsValidPdf())
                    return Results.BadRequest("
                        Invalid PDF format");

                // Submit for processing
                var jobId = await documentService.
                    SubmitForProcessingAsync(
                    document);

                // Return tracking ID
                return Results.Accepted("/api/
                    documents/status/" + jobId,
                    jobId);
            }
            catch (Exception ex) {
                return Results.Problem(ex.Message)
                    ;
            }
        })
        .RequireAuthorization()
        .WithOpenApi();
    }
}
```

The authorization model is simplified to:

$$A(u, r) = \begin{cases} 1, & \text{if user } u \text{ authorized for resource } r \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

2) Document Processing Service: This microservice orchestrates the document analysis workflow. It determines document type, routes to appropriate pipelines, and manages the processing state machine.

Listing 2: Document Processing Pipeline

```
public class DocumentProcessingService :
    IDocumentProcessingService
{
    private readonly IDocumentRepository
        _repository;
    private readonly IMessageBus _messageBus;
    private readonly ILogger<
        DocumentProcessingService> _logger;

    public async Task<ProcessingResult>
        ProcessDocumentAsync(DocumentId documentId
        )
    {
        var document = await _repository.
            GetDocumentAsync(documentId);

        // Determine document type
        var documentType = await _messageBus.
            SendAsync<DocumentType>(new
            ClassifyDocumentCommand(document));

        // Process based on document type
        ProcessingResult result;
        if (documentType == DocumentType.
            DigitalPdf) {
            result = await _messageBus.SendAsync<
                ProcessingResult>(new
                ProcessDigitalPdfCommand(document)
                );
        }
        else if (documentType == DocumentType.
            ScannedDocument) {
            result = await _messageBus.SendAsync<
                ProcessingResult>(new
                ProcessScannedDocumentCommand(
                document));
        }
        else {
            throw new
                UnsupportedDocumentTypeException(
                documentType);
        }

        // Extract clauses
        var clauses = await _messageBus.SendAsync<
            IEnumerable<Clause>>(new
            ExtractClausesCommand(result.Content))
            ;

        // Generate embeddings
        await _messageBus.PublishAsync(new
            GenerateEmbeddingsCommand(clauses));

        return new ProcessingResult(documentId,
            ProcessingStatus.Completed);
    }
}
```

3) Embedding Service: This service converts document clauses into semantic vector embeddings. It pre-processes text, calls the Qwen embedding model, and normalizes the resulting vectors.

Listing 3: Embedding Generation

```
public class EmbeddingService : IEmbeddingService
{
    private readonly IQwenModelClient _modelClient
        ;
    private readonly IVectorRepository
        _vectorRepository;

    public async Task<IReadOnlyList<
        ClauseEmbedding>> GenerateEmbeddingsAsync(
        IReadOnlyList<DocumentClause> clauses)
    {
        // Preprocess clauses for optimal
            embedding
        var preprocessedClauses = new List<
            PreprocessedClause>();
        foreach (var c in clauses) {
            preprocessedClauses.Add(new
                PreprocessedClause(
                c.Id,
                TextPreprocessor.Normalize(c.Text)
                    ,
                c.Metadata));
        }

        // Generate embeddings in batches
        var embeddings = new List<ClauseEmbedding
            >();
        foreach (var batch in BatchList(
            preprocessedClauses, 32))
        {
            // Extract text for batch
            var batchTexts = new List<string>();
            foreach (var c in batch) {
                batchTexts.Add(c.Text);
            }

            // Call Qwen model to generate
                embeddings
            var embeddingResults = await
                _modelClient.
                GenerateEmbeddingsAsync(batchTexts
                );

            // Process and normalize results
            for (int i = 0; i < batch.Count; i++)
            {
                var clause = batch[i];
                var vector = embeddingResults[i];

                // Normalize vector to unit length
                var normalizedVector =
                    NormalizeVector(vector);

                embeddings.Add(new ClauseEmbedding
                    (
                    clause.Id,
                    normalizedVector,
                    clause.Metadata));
            }
        }

        // Store embeddings in vector database
        await _vectorRepository.
            StoreEmbeddingsAsync(embeddings);

        return embeddings;
```

```
    }

    private float[] NormalizeVector(float[] vector)
    {
        float magnitude = (float)Math.Sqrt(vector.Sum(x => x * x));
        float[] normalized = new float[vector.Length];
        for (int i = 0; i < vector.Length; i++) {
            normalized[i] = vector[i] / magnitude;
        }
        return normalized;
    }

    private List<List<T>> BatchList<T>(List<T> source, int batchSize)
    {
        var result = new List<List<T>>();
        for (int i = 0; i < source.Count; i += batchSize)
        {
            result.Add(source.GetRange(i, Math.Min(batchSize, source.Count - i)));
        }
        return result;
    }
}
```

4) Fraud Detection Service: The core analytical engine performs vector similarity searches and applies threshold-based detection with the simplified decision function:

$$F(c, c_r) = \begin{cases} 1 & \text{if } sim(c, c_r) < \tau_1 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Listing 4: Fraud Detection Implementation

```
public class FraudDetectionService : IFraudDetectionService
{
    private readonly IVectorRepository _vectorRepository;
    private readonly ILlmReasoningService _llmService;
    private readonly IOptions<FraudDetectionOptions> _options;

    public async Task<DetectionResult> AnalyzeDocumentAsync(DocumentId documentId)
    {
        // Get document clauses and their embeddings
        var clauseEmbeddings = await _vectorRepository.GetDocumentEmbeddingsAsync(documentId);
        var detectedFrauds = new List<FraudAlertInfo>();

        // Analyze each clause
        foreach (var clauseEmbedding in clauseEmbeddings)
        {
            // Find similar clauses in the reference corpus
            var similarClauses = await _vectorRepository.FindSimilarClausesAsync(
                clauseEmbedding,
                _options.Value.SimilarityThreshold,
                _options.Value.MaxResults);

            if (similarClauses.Count == 0)
                continue;

            // Check if similarity score indicates potential fraud
            var mostSimilarClause = similarClauses[0];
            if (mostSimilarClause.SimilarityScore < _options.Value.FraudThreshold)
            {
                // Get original clause texts
                var originalClause = await _vectorRepository.GetClauseTextAsync(mostSimilarClause.ClauseId);
                var suspiciousClause = await _vectorRepository.GetClauseTextAsync(clauseEmbedding.ClauseId);

                // Send to LLM for detailed analysis
                var llmAnalysis = await _llmService.AnalyzeClauseDifferencesAsync(originalClause, suspiciousClause);

                // Create alert if LLM indicates significant changes
                if (llmAnalysis.Significance > _options.Value.LlmSignificanceThreshold)
                {
                    detectedFrauds.Add(new FraudAlertInfo(clauseEmbedding.ClauseId, mostSimilarClause.ClauseId, mostSimilarClause.SimilarityScore, llmAnalysis));
                }
            }
        }

        return new DetectionResult(documentId, detectedFrauds);
    }
}
```

D. Resilience and Scalability Patterns

The architecture incorporates circuit breakers, retry policies, and bulkhead isolation with the circuit breaker state machine:

$$CB = \{Closed, HalfOpen, Open\} \tag{6}$$

Listing 5: Resilience Implementation

```
public static class ResilienceExtensions
{
    public static IServiceCollection AddResiliencePolicies(
        this IServiceCollection services)
    {
```

```
        // Add HTTP client with circuit breaker and retry patterns
        services.AddHttpClient("embedding-service", client =>
        {
            client.BaseAddress = new Uri("http://embedding-service/");
            client.Timeout = TimeSpan.FromSeconds(30);
        })
        .AddResilienceHandler("circuit-breaker", builder =>
        {
            builder.AddCircuitBreaker(options =>
            {
                options.SamplingDuration = TimeSpan.FromSeconds(30);
                options.FailureRatio = 0.2;
                options.MinimumThroughput = 5;
                options.BreakDuration = TimeSpan.FromSeconds(30);
                options.OnOpened = () =>
                    Log.Warning("Circuit opened for 30s");
                options.OnClosed = () =>
                    Log.Information("Circuit closed");
                options.OnHalfOpened = () =>
                    Log.Information("Circuit half-open");
            });
        })
        .AddResilienceHandler("retry", builder =>
        {
            builder.AddRetry(options =>
            {
                options.MaxRetryAttempts = 3;
                options.Delay = TimeSpan.FromSeconds(1);
                options.UseJitter = true;
                options.BackoffType = DelayBackoffType.Exponential;
                options.OnRetry = (outcome, delay, attempt, context) =>
                    Log.Warning("Retry {0} after {1}ms",
                        attempt, delay.TotalMilliseconds);
            });
        });

        return services;
    }
}
```

This architecture represents a theoretical design that leverages modern .NET Core 9 capabilities while remaining implementable with current open-source technologies. The modular approach allows for incremental implementation and testing, with each component providing value even before the complete system is deployed.

IV. Document Processing Pipeline

At the heart of our fraud detection system is a sophisticated document processing pipeline that transforms raw legal documents into structured, analyzable content. This section details the technical approach to document ingestion, parsing, and segmentation.

A. Mathematical Model of the Document Processing Pipeline

We model the document processing pipeline as a series of transformation functions that progressively refine the representation of a document. Let D_{raw} represent a raw document in its initial form (PDF file).

$$D_{processed} = f_n \circ f_{n-1} \circ \ldots \circ f_1(D_{raw}) \tag{7}$$

$$D_{text} = \begin{cases} f_{digital}(D_{raw}), & \text{if digital PDF} \\ f_{OCR}(D_{raw}), & \text{if scanned document} \end{cases} \tag{8}$$

These paths converge after text extraction, allowing subsequent processing stages to operate on a unified text representation.

B. Document Ingestion

The document ingestion process begins with the receipt of legal documents through the API Gateway. Documents are accepted in multiple formats, with PDF being the primary supported format given its prevalence in legal contexts. The system performs initial validation checks:

- File integrity verification
- Virus/malware scanning
- Format validation
- Basic metadata extraction (file creation date, modification date)
- Document type classification (scanned vs. digital)

Valid documents receive a unique identifier and are stored in the secure document storage service with appropriate metadata tags. The document is then queued for processing based on its classification.

The classification function C maps a document to a document type:

$$C : D_{raw} \rightarrow \{digital, scanned\} \times \{contract, deed, will, \ldots\} \tag{9}$$

C. Digital PDF Processing

For text-based PDFs (those containing actual text data rather than just images), the system employs open-source .NET libraries for content extraction:

- PdfPig provides low-level access to PDF structure, enabling precise text location and formatting details to be preserved
- PdfSharpCore offers an alternative with strong support for complex document layouts

The extraction process carefully preserves:

- Text content with original formatting
- Structural elements like headings, paragraphs, and lists
- Table structures and their content
- Page numbers and section markers
- Document outline/bookmarks if present

This structured extraction is critical for maintaining the logical organization of the document, which aids in subsequent clause identification and comparison.

The digital PDF processing function $f_{digital}$ can be formalized as:

$$f_{digital}(D_{raw}) = (T, S, L) \tag{10}$$

where T represents the extracted text content, S represents the structural elements, and L represents the layout information.

D. Scanned Document OCR Processing

For scanned documents or image-based PDFs, the OCR Worker component performs text extraction:

- Pages are extracted and pre-processed to enhance image quality
- Deskewing, denoising, and contrast enhancement techniques are applied
- Tesseract OCR (via the TesseractSharp .NET binding) processes each page
- Language-specific models are applied based on detected content
- Post-OCR correction algorithms address common recognition errors

The OCR process is configured with legal document-specific settings, including:

- Custom dictionaries of legal terminology
- Recognition patterns for common legal document structures
- Page segmentation modes optimized for multi-column legal texts

To ensure high accuracy, the OCR output includes confidence scores for each recognized word, allowing downstream components to handle uncertain text appropriately.

Mathematically, the OCR function f_{OCR} maps a document to extracted text with confidence metrics:

$$f_{OCR}(D_{raw}) = (T, C, \hat{S}) \tag{11}$$

where T is the extracted text, C is a matrix of confidence scores, and $\hat{S}$ is an estimated structure inferred from the layout analysis.

The confidence matrix C contains elements c_{ij} representing the confidence score for the j-th character in the i-th word, where $c_{ij} \in [0, 1]$.

E. OCR Confidence Model

For OCR-processed text, the system maintains confidence scores that influence downstream analysis. If a legal term has low OCR confidence, it may be flagged for special attention during semantic comparison, as differences could be OCR errors rather than actual modifications.

The confidence matrix C for a document with m words can be represented as:

$$C = \begin{bmatrix} c_{1,1} & c_{1,2} & \dots & c_{1,n_1} \\ c_{2,1} & c_{2,2} & \dots & c_{2,n_2} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m,1} & c_{m,2} & \dots & c_{m,n_m} \end{bmatrix} \tag{12}$$

where $c_{i,j}$ is the confidence score for the j-th character in the i-th word, and n_i is the length of the i-th word.

Listing 6: OCR Result JSON Structure

```
{
  "document_id": "d8e7c3b2-a1f9-4e5d-b6c7-
      d8e9f0a1b2c3",
  "page_count": 12,
  "pages": [
    {
      "page_number": 1,
      "words": [
        {
          "text": "Agreement",
          "bounding_box": [120, 240, 200, 265],
          "confidence": 0.98,
          "characters": [
            {"char": "A", "confidence": 0.99},
            {"char": "g", "confidence": 0.98},
            {"char": "r", "confidence": 0.97},
            {"char": "e", "confidence": 0.99},
            {"char": "e", "confidence": 0.99},
            {"char": "m", "confidence": 0.97},
            {"char": "e", "confidence": 0.98},
            {"char": "n", "confidence": 0.97},
            {"char": "t", "confidence": 0.99}
          ]
        },
        // Additional words...
      ],
      "paragraphs": [
        {
          "start_word_index": 0,
          "end_word_index": 42,
          "confidence": 0.95,
          "is_heading": true
        }
        // Additional paragraphs...
      ]
    }
    // Additional pages...
  ]
}
```

F. Document Structure Analysis

Once the raw text is extracted (either directly from digital PDFs or via OCR), the system performs document structure analysis:

- Document type identification (contract, deed, will, etc.)
- Section and subsection boundary detection
- Paragraph and sentence splitting
- Legal clause identification
- Reference and cross-reference detection
- Numbered list and bullet point recognition

This structural analysis employs a combination of rule-based patterns specific to legal documents and machine learning models trained on legal document corpora. The result is a

hierarchical representation of the document that preserves its logical organization.

The structure analysis function $f_{structure}$ can be defined as:

$$f_{structure}(T, S) = H \tag{13}$$

where H is a hierarchical representation of the document structure, typically modeled as a tree where:

$$H = (V, E, \lambda) \tag{14}$$

V is the set of nodes (sections, paragraphs, clauses), E is the set of edges representing the hierarchical relationships, and λ is a labeling function that assigns types and attributes to each node.

G. Legal Clause Segmentation

Perhaps the most critical step in the pipeline is the identification and segmentation of individual legal clauses. Legal clauses represent distinct obligations, rights, or conditions within a contract and are the primary unit of analysis for fraud detection.

The system identifies clauses through:

- Explicit clause numbering and headers
- Linguistic markers common in legal language ("shall," "must," "herein")
- Sentence and paragraph structure analysis
- Semantic boundary detection using embeddings

For complex documents without explicit clause numbering, the system uses a sliding window approach with overlap to ensure no critical content is missed. Each identified clause is assigned a unique identifier and stored with:

- The clause text
- Position within the document (page, paragraph)
- Hierarchical location (section, subsection)
- Type classification (e.g., indemnity, termination, payment)
- Detected entities (parties, dates, amounts)
- Confidence score of the segmentation

The clause segmentation function f_{clause} can be formalized as:

$$f_{clause}(H) = \{(c_1, m_1, \gamma_1), (c_2, m_2, \gamma_2), \ldots, (c_k, m_k, \gamma_k)\} \tag{15}$$

where c_i is the text of the i-th clause, m_i is the metadata associated with the clause, and $\gamma_i \in [0, 1]$ is the confidence score of the segmentation.

H. Preprocessing for Semantic Analysis

Before the clauses proceed to embedding generation, several preprocessing steps ensure optimal semantic analysis:

- Normalization of formatting variations
- Resolution of abbreviations and legal shorthand
- Standardization of dates, currencies, and measurements
- Noise removal (e.g., page numbers, headers in extracted text)
- Named entity normalization to reduce spurious differences

These preprocessing steps help focus the subsequent semantic analysis on substantive differences rather than superficial variations.

The preprocessing function $f_{preprocess}$ can be defined as:

$$f_{preprocess}(C) = \{(c_i', m_i, \gamma_i) \mid (c_i, m_i, \gamma_i) \in C\} \tag{16}$$

where C is the set of all clauses with their metadata and confidence scores, and c_i' is the normalized version of clause text c_i.

The complete preprocessing transformation can be expressed as:

$$f_{preprocess}(\{(c_i, m_i, \gamma_i)\}_{i=1}^{k}) = \{(c_i', m_i, \gamma_i)\}_{i=1}^{k} \tag{17}$$

I. Pipeline Orchestration

The entire document processing pipeline is orchestrated using a combination of:

- Background worker services for long-running processes
- Message queues for reliable handoff between stages
- State persistence for tracking processing status
- Error handling with appropriate retry policies

Wolverine provides in-process messaging with built-in command and event handling, while more durable message queues like RabbitMQ or Azure Service Bus handle cross-service communication. The pipeline implementation follows the Pipes and Filters pattern, allowing for flexible insertion of additional processing steps as requirements evolve.

The orchestration can be modeled as a directed acyclic graph (DAG) $G = (V, E)$ where vertices represent processing tasks and edges represent dependencies and data flow. The critical path through this graph determines the minimum processing time for a document:

$$T_{min} = \sum_{v \in CP} t(v) \tag{18}$$

where CP is the critical path through G and $t(v)$ is the processing time for task v.

The resulting processed document, with its hierarchical structure and segmented clauses, serves as the foundation for the semantic embedding and fraud detection processes described in subsequent sections.

V. Semantic Embedding and Similarity Matching

Once legal documents are processed and clauses are identified, the next critical step is converting these text segments into vector representations that capture their semantic meaning. This section details the embedding generation process and the similarity matching techniques used to identify potentially fraudulent alterations.

A. Embedding Model Selection

For our theoretical system, we've selected the open-source Qwen family of models, specifically for their strong performance on semantic text representation tasks. The key advantages of this choice include:

- Open-source availability, allowing for local deployment without API dependencies
- Strong multilingual capabilities, important for international legal documents
- Competitive performance with closed-source alternatives
- Availability of different model sizes to balance accuracy and computational requirements
- ONNX runtime compatibility for optimized inference

Within the Qwen family, two primary options are considered:

- **qwen-embedding**: A dedicated embedding model optimized specifically for vector representation
- **qwen2.5-7b-instruct**: A larger instruction-tuned model that can generate embeddings as well as perform reasoning tasks

The embedding service is designed to support either model, with the specific choice depending on deployment constraints and performance requirements.

B. Embedding Generation Process

The embedding generation process follows these steps:

1) Preprocessed clause text is tokenized using the model-specific tokenizer
2) For clauses exceeding the model's context window, a sliding window approach with overlap is used
3) The model generates fixed-length vector representations (typically 768 or 1024 dimensions)
4) For clauses processed with multiple windows, a weighted averaging technique combines the vectors
5) The final vectors are L2-normalized to enable cosine similarity comparisons

To optimize performance, the embedding service implements batching, allowing multiple clauses to be processed in a single model inference pass. This significantly improves throughput, especially on GPU hardware.

C. Trusted Corpus Management

A critical component of the fraud detection system is the trusted corpus: a collection of verified, authentic legal documents that serve as the reference point for detecting alterations. The system maintains:

- Standard contract templates organized by document type and jurisdiction
- Historical versions of documents with clear version control
- Common clause libraries with metadata about their typical usage and intent
- Embeddings of all clauses in the trusted corpus, precomputed and indexed

The trusted corpus is maintained in a controlled environment with strict governance controls. Updates to the corpus follow a formal review process to ensure only verified authentic documents are included.

D. Vector Database Implementation

The Qdrant vector database stores and indexes the embeddings for efficient similarity search. Key features of the implementation include:

- Collections organized by document type and jurisdiction
- Metadata filters to narrow search scope based on clause type and document context
- Versioned storage to track changes to reference documents over time
- Optimized indexing using HNSW (Hierarchical Navigable Small World) algorithm
- Payload storage for clause text and metadata alongside vectors

The database schema includes:

- Vector ID: A unique identifier for each clause embedding
- Vector: The normalized embedding representing the clause semantics
- Document ID: Reference to the source document
- Clause ID: Identifier of the specific clause within the document
- Clause Text: The original text for reference and display
- Clause Type: Classification of the clause (e.g., indemnity, termination)
- Version: Document version information
- Timestamp: When the vector was added to the database
- Confidence: OCR confidence or other quality metrics

E. Similarity Search Algorithm

When a new document is analyzed, each extracted clause undergoes similarity search against the trusted corpus:

1) The clause embedding is generated as described earlier
2) Initial filtering narrows the search space based on document type and clause classification
3) The filtered vector database is searched for the k-nearest neighbors to the query embedding
4) Similarity scores (cosine similarity) are computed for each match
5) Matches above a high similarity threshold (e.g., 0.95) are considered potential direct matches
6) Matches in a middle range (e.g., 0.80-0.95) are flagged for closer inspection as potential alterations
7) Matches below a minimum threshold are considered unrelated clauses

The search parameters are adjustable based on the specific requirements and risk tolerance of the application. Higher precision settings may be used for critical clauses (such as indemnity or liability clauses) to ensure even minor alterations are detected.

F. Detecting Semantic Alterations

The core challenge in legal document fraud detection is distinguishing between innocuous variations (formatting, minor wording changes that preserve meaning) and substantive alterations that change legal implications. Several techniques are employed:

- **Differential Analysis**: Comparing similarity scores across different sections of a clause to identify localized alterations
- **Contextual Embeddings**: Using position-aware embeddings to capture how meaning changes based on surrounding content
- **Entity-Aware Comparison**: Special handling of named entities, dates, amounts, and other critical elements
- **Legal Term Sensitivity**: Heightened scrutiny for specialized legal terminology with precise meanings

G. Handling Ambiguous Cases

Not all deviations from reference documents represent fraud. Legal documents are often legitimately customized for specific situations. The system handles ambiguity through:

- Confidence scoring that reflects the certainty of a potential issue
- Explicit flagging of ambiguous cases for human review
- LLM-powered reasoning to explain potential implications (detailed in the next section)
- Progressive refinement of detection thresholds based on feedback

H. Performance Considerations

Vector similarity search at scale presents performance challenges, particularly for large document collections. The implementation addresses these through:

- Efficient indexing structures (HNSW) to enable sub-linear search complexity
- Strategic filtering to reduce the search space before vector comparison
- Parallel processing of multiple clauses across available compute resources
- Caching of frequently accessed reference vectors
- Progressive search refinement, starting with broad clustering before detailed comparison

The semantic embedding and similarity matching components form the technical core of the fraud detection system, translating natural language legal content into mathematical representations that enable precise comparison and alteration detection. This approach bridges the gap between traditional text comparison methods and the nuanced understanding required for legal document analysis.

VI. LLM Clause Reasoning

A key component of our document fraud detection system is the ability to analyze and explain subtle but semantically significant changes in legal clauses. This section describes how we integrate local Large Language Models (LLMs) to provide natural language reasoning about detected discrepancies.

A. Semantic Reasoning Architecture

The LLM reasoning component provides several critical functions:

- Detailed comparison of original and potentially modified clauses
- Analysis of how modifications affect legal rights and obligations
- Identification of beneficiaries of clause modifications
- Assessment of significance level for detected changes
- Explanation of potential implications of detected alterations
- Generating natural language summaries of findings for human reviewers
- Suggesting possible intentions behind identified modifications

Listing 7: LLM Reasoning Service

```
public class LlmReasoningService :
    ILlmReasoningService
{
    private readonly ILlmClient _llmClient;
    private readonly IOptions<LlmOptions> _options
        ;
    private readonly ILogger<LlmReasoningService>
        _logger;

    public async Task<ClauseAnalysisResult>
        AnalyzeClauseDifferencesAsync(
        string originalClause,
        string modifiedClause,
        ClauseType clauseType,
        string documentContext)
    {
        try
        {
            // Prepare the prompt
            var prompt = BuildPrompt(
                originalClause,
                modifiedClause,
                clauseType,
                documentContext);

            // Call the LLM
            var response = await _llmClient.
                GenerateCompletionAsync(
                new LlmRequest
                {
                    SystemPrompt = _options.Value.
                        SystemPrompt,
                    UserPrompt = prompt,
                    MaxTokens = _options.Value.
                        MaxTokens,
                    Temperature = 0.2, // Low
                        temperature for consistent
                         analysis
                    ResponseFormat = "json"
                });

            // Parse the response
            var analysis = JsonSerializer.
                Deserialize<ClauseAnalysis>(
                response.Completion,
                new JsonSerializerOptions
                {
                    PropertyNameCaseInsensitive =
                        true
                });
```

```
            return new ClauseAnalysisResult
            {
                TextDifferences = analysis.
                    TextDifferences,
                LegalAnalysis = analysis.
                    LegalAnalysis,
                Beneficiary = analysis.Beneficiary
                    ,
                Significance = MapSignificance(
                    analysis.Significance),
                Explanation = analysis.Explanation
                    ,
                Confidence = analysis.Confidence
            };
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Error analyzing
                clause differences");
            throw;
        }
    }

    private string BuildPrompt(
        string originalClause,
        string modifiedClause,
        ClauseType clauseType,
        string documentContext)
    {
        return $@"Original clause: ""{
            originalClause}""

Modified clause: ""{modifiedClause}""

Clause type: {clauseType}
Document context: {documentContext}

Please analyze:
1. What specific words or phrases were changed?
2. Do these changes alter legal rights,
    obligations, or liabilities? How?
3. Which party might benefit from these changes?
4. Rate the significance of the change (minor/
    moderate/major)
5. Provide a brief explanation in plain language
    that a non-lawyer could understand.";
    }

    private SignificanceLevel MapSignificance(
        string significance)
    {
        return significance.ToLowerInvariant()
            switch
        {
            "minor" => SignificanceLevel.Minor,
            "moderate" => SignificanceLevel.
                Moderate,
            "major" => SignificanceLevel.Major,
            _ => SignificanceLevel.Unknown
        };
    }
}
```

B. Mathematical Framework for Semantic Difference Analysis

We can formalize the LLM reasoning process using a mathematical framework. Let C_{orig} and C_{mod} represent the original and modified clauses, respectively. The semantic difference function Δ_S can be defined as:

$$\Delta_S(C_{orig}, C_{mod}) = \{(r_1, w_1), (r_2, w_2), ..., (r_n, w_n)\} \quad (19)$$

where each tuple (r_i, w_i) represents a semantic relationship r_i that has changed with significance weight $w_i \in [0, 1]$.

The set of possible semantic relationships includes key legal concepts:

$$R = \{r_1, r_2, ..., r_8\} \quad (20)$$

where r_1 = obligation, r_2 = right, r_3 = liability, r_4 = condition, r_5 = privilege, r_6 = power, r_7 = immunity, and r_8 = disability.

The overall semantic change significance S can be quantified as:

$$S(C_{orig}, C_{mod}) = \sum_{i=1}^{n} w_i \cdot I(r_i) \quad (21)$$

where $I(r_i)$ is an importance function that maps semantic relationship types to their relative importance in legal contexts.

The LLM implements an approximation of this theoretical framework, providing both structured analysis and natural language explanation of the semantic differences.

C. Model Selection and Deployment

For our theoretical system, we propose using the open-source Qwen2.5-7b-instruct model or a similar alternative. Key considerations in this selection include:

- Open-source availability for on-premises deployment
- Sufficient reasoning capabilities for legal text analysis
- Manageable computational requirements for enterprise deployment
- Support for efficient inference via ONNX Runtime or similar frameworks

The model is deployed as a containerized service accessible to the fraud detection component. For organizations with more computational resources, larger models like Qwen2-72b-instruct could provide enhanced reasoning capabilities at the cost of increased hardware requirements.

Listing 8: LLM Client Implementation

```
public class QwenLlmClient : ILlmClient
{
    private readonly IOnnxModelService
        _modelService;
    private readonly ITokenizer _tokenizer;
    private readonly IOptions<QwenOptions>
        _options;

    public async Task<LlmResponse>
        GenerateCompletionAsync(LlmRequest request
        )
    {
        // Tokenize the input
        var systemTokens = _tokenizer.Encode(
            _options.Value.SystemPrefix + request.
            SystemPrompt);
        var userTokens = _tokenizer.Encode(
            _options.Value.UserPrefix + request.
            UserPrompt);
        var assistantTokens = _tokenizer.Encode(
            _options.Value.AssistantPrefix);

        // Combine tokens
```

```
        var inputTokens = new List<int>();
        inputTokens.AddRange(systemTokens);
        inputTokens.AddRange(userTokens);
        inputTokens.AddRange(assistantTokens);

        // Prepare input tensor
        var inputTensor = new DenseTensor<long>(
            inputTokens.Select(t => (long)t).
                ToArray(),
            new[] { 1, inputTokens.Count });

        // Run inference
        var outputTokens = await _modelService.
            RunInferenceAsync(
            inputTensor,
            request.MaxTokens,
            request.Temperature);

        // Decode output
        var outputText = _tokenizer.Decode(
            outputTokens);

        return new LlmResponse
        {
            Completion = outputText,
            TokensUsed = inputTokens.Count +
                outputTokens.Length
        };
    }
}

// ONNX Runtime integration for efficient
    inference
public class OnnxModelService : IOnnxModelService
{
    private readonly string _modelPath;
    private readonly SemaphoreSlim _semaphore;
    private InferenceSession _session;

    public OnnxModelService(IOptions<QwenOptions>
        options)
    {
        _modelPath = options.Value.ModelPath;
        _semaphore = new SemaphoreSlim(1, 1); //
            Control concurrent access
    }

    public async Task<int[]> RunInferenceAsync(
        Tensor<long> inputIds,
        int maxNewTokens,
        float temperature)
    {
        await EnsureSessionInitializedAsync();

        await _semaphore.WaitAsync();
        try
        {
            var inputs = new Dictionary<string,
                OrtValue>
            {
                ["input_ids"] = OrtValue.
                    CreateTensorValueFromMemory(
                    inputIds.Buffer, inputIds.
                    Dimensions)
            };

            var runOptions = new RunOptions();
            runOptions.Add("max_length", inputIds.
                Length + maxNewTokens);
            runOptions.Add("temperature",
                temperature);

            using var outputs = await _session.
                RunAsync(runOptions, inputs);

            var outputTensor = outputs["output_ids
                "].GetTensorDataAsSpan<int>().
                ToArray();

            // Extract only the newly generated
                tokens (remove input prefix)
            return outputTensor.Skip(inputIds.
                Length).ToArray();
        }
        finally
        {
            _semaphore.Release();
        }
    }

    private async Task
        EnsureSessionInitializedAsync()
    {
        if (_session == null)
        {
            await _semaphore.WaitAsync();
            try
            {
                if (_session == null)
                {
                    var sessionOptions = new
                        SessionOptions();
                    sessionOptions.
                        EnableMemoryPattern =
                        false;
                    sessionOptions.ExecutionMode =
                         ExecutionMode.
                        ORT_SEQUENTIAL;

                    // Add GPU support if
                        available
                    var gpu = true; // Example
                        flag
                    if(gpu)
                    {
                sessionOptions.
                    AppendExecutionProvider_CUDA
                    ();
                    }

                    _session = new
                        InferenceSession(
                        _modelPath, sessionOptions
                        );
                }
            }
            finally
            {
                _semaphore.Release();
            }
        }
    }
}
```

D. Prompt Engineering for Legal Analysis

The effectiveness of the LLM reasoning component heavily depends on careful prompt engineering. Our system uses structured prompts that guide the model to perform specific types of analysis:

Listing 9: Prompt Configuration

```
public class LlmPromptConfiguration
{
    public const string SystemPrompt = @"
```

```
You are a specialized legal analysis AI focusing on document fraud detection.
Your task is to analyze differences between original and modified legal clauses.
Consider the legal implications of any changes, focusing on:
- Rights, obligations, and liabilities
- Which party benefits from the changes
- The significance of the changes
- Clear explanations for non-lawyers

Provide your analysis in JSON format with the following structure:
{
  ""textDifferences"": ""Detailed list of specific words/phrases changed"",
  ""legalAnalysis"": ""Analysis of how changes affect legal rights/obligations"",
  ""beneficiary"": ""Party that benefits from these changes"",
  ""significance"": ""minor|moderate|major"",
  ""explanation"": ""Clear explanation for non-lawyers"",
  ""confidence"": 0.0-1.0
}";

    public static string BuildComparisonPrompt(
        string original,
        string modified,
        string clauseType)
    {
        return $@"
ORIGINAL CLAUSE:
""{original}""

MODIFIED CLAUSE:
""{modified}""

CLAUSE TYPE: {clauseType}

Analyze the differences between these clauses, focusing on legal implications.";
    }
}
```

E. Integration with Document Processing Pipeline

The LLM reasoning service integrates into the overall document processing pipeline through well-defined interfaces and asynchronous processing:

Listing 10: Integration Configuration

```
public static class LlmReasoningServiceExtensions
{
    public static IServiceCollection AddLlmReasoningServices(
        this IServiceCollection services,
        IConfiguration configuration)
    {
        // Register configuration
        services.Configure<LlmOptions>(
            configuration.GetSection("LlmOptions"));

        services.Configure<QwenOptions>(
            configuration.GetSection("QwenOptions"));

        // Register services
        services.AddSingleton<ITokenizer,
            QwenTokenizer>();
        services.AddSingleton<IOnnxModelService,
            OnnxModelService>();
        services.AddSingleton<ILlmClient,
            QwenLlmClient>();
        services.AddScoped<ILlmReasoningService,
            LlmReasoningService>();

        // Register background services
        services.AddHostedService<
            LlmModelPrewarmService>();

        return services;
    }
}
```

F. Performance Optimization

To ensure optimal performance in a production environment, the LLM reasoning component implements several optimization strategies:

- Model quantization to 4-bit or 8-bit precision to reduce memory requirements
- Batched processing of clause comparisons when possible
- Caching of analysis results for identical or highly similar clause pairs
- Asynchronous processing with priority queueing for critical document analysis
- Optimized tensor operations using ONNX Runtime or similar acceleration frameworks

Listing 11: Model Optimization

```
public class ModelOptimizationService :
    IModelOptimizationService
{
    private readonly IOptions<OptimizationOptions>
        _options;
    private readonly ILogger<
        ModelOptimizationService> _logger;

    public async Task<string> OptimizeModelAsync(
        string modelPath)
    {
        _logger.LogInformation("Optimizing model
            at {ModelPath}", modelPath);

        var outputPath = Path.Combine(
            Path.GetDirectoryName(modelPath),
            Path.GetFileNameWithoutExtension(
                modelPath) + "_optimized.onnx");

        // Create optimization options
        var sessionOptions = new SessionOptions();
        sessionOptions.GraphOptimizationLevel =
            GraphOptimizationLevel.ORT_ENABLE_ALL;
        sessionOptions.EnableMemoryPattern = false
            ;

        // Quantization options
        var quantizationOptions = new
            QuantizationOptions
        {
            QuantizationAlgorithm =
                QuantizationAlgorithm.QDQ,
            QuantizationGranularity =
                QuantizationGranularity.PerTensor,
            QuantizationDataType =
                QuantizationDataType.QInt8
        };
```

```
        // Optimize and save model
        using var session = new InferenceSession(
            modelPath, sessionOptions);
        var optimizedModelBytes = session.
            OptimizeModel(
            outputPath,
            "CPU",
            enableQuantization: _options.Value.
                EnableQuantization,
            quantizationOptions:
                quantizationOptions);

        _logger.LogInformation("Model optimized
            and saved to {OutputPath}", outputPath
            );

        return outputPath;
    }
}
```

This LLM reasoning component provides the human-understandable analysis that makes our document fraud detection system not just a binary classifier, but an explanatory tool that helps legal professionals understand the nature, significance, and potential implications of detected clause modifications.

VII. Case Study: Altered Indemnity Clause in Real Estate Sale

Let's dive into a real-world example to show what our system can actually do. Here's a hypothetical case that demonstrates how our document fraud detection pipeline would catch and analyze a sneakily altered indemnity clause in a big-ticket real estate deal. This example really highlights how the system spots semantic manipulation that could cost someone serious money [20].

A. Scenario Overview

Picture this: there's a commercial property on the market for $4.8 million. The standard purchase agreement (you know, the template everyone uses in that jurisdiction) has a pretty standard indemnity clause that shields the buyer from certain liabilities stemming from stuff the seller did before the sale [3].

Now here's where it gets interesting. During the deal process, someone slips in a digitally modified version of the contract. It looks identical to the template - same signatures, same formatting, the works. But there's a catch - a critical indemnity clause has been subtly tweaked to push liability back onto the buyer in specific situations. We're talking about a potential $1.2 million exposure based on known environmental cleanup costs that would need to be addressed [2].

B. Original vs. Modified Clauses

The original clause from the trusted reference template reads:

> **7.3 Indemnification.** Seller agrees to indemnify, defend, and hold harmless Buyer from and against any and all claims, liabilities, losses, damages, costs, and expenses, including reasonable attorney's fees, arising from or relating to any act, omission, negligence, or misconduct of Seller occurring prior to the Closing Date, including but not limited to any undisclosed defects in the Property known to Seller at the time of sale.

The modified clause in the submitted document reads:

> **7.3 Indemnification.** Seller agrees to indemnify, defend, and hold harmless Buyer from and against any and all claims, liabilities, losses, damages, costs, and expenses, including reasonable attorney's fees, arising from or relating to any act, omission, negligence, or misconduct of Seller occurring prior to the Closing Date, except for any undisclosed defects in the Property, which shall be the sole responsibility of Buyer upon taking possession.

Did you catch the change? It's subtle but huge: they switched "including but not limited to any undisclosed defects" to "except for any undisclosed defects." That completely flips who's on the hook for hidden problems with the property.

C. Mathematical Risk Assessment Model

We can formalize the financial risk exposure using a mathematical model that quantifies the impact of this semantic change [21]:

Let R_B represent the buyer's risk exposure, R_S represent the seller's risk exposure, and C represent the total potential remediation costs. In the original clause:

$$R_B = 0 \tag{22}$$
$$R_S = C \tag{23}$$

With the modified clause:

$$R'_B = C \tag{24}$$
$$R'_S = 0 \tag{25}$$

The financial risk transfer function ΔR can be defined as:

$$\Delta R = R'_B - R_B = C \tag{26}$$

In our scenario, with $C = \$1.2$ million in known environmental remediation costs, the clause modification represents a $\Delta R = \$1.2$ million shift in financial liability from the seller to the buyer.

The probability of detection P_d using traditional methods versus our semantic approach can be modeled as [18]:

$$P_d(\text{traditional}) = \frac{N_{\text{changed chars}}}{N_{\text{total chars}}} \approx 0.15 \tag{27}$$
$$P_d(\text{semantic}) = f(v_s, \lambda) \approx 0.98 \tag{28}$$

Where v_s represents vector similarity score and λ represents the LLM analysis confidence parameter.

D. Detection Process

Let's walk through how our system would process this document and detect the fraudulent alteration:

1) Document Ingestion and Processing: The digitally modified PDF is uploaded to the system and initially validated. Since it's a text-based PDF rather than a scanned document, the PdfPig library [5] extracts the textual content while preserving structure and formatting.

2) Document Structure Analysis: The system identifies the document as a commercial real estate purchase agreement based on its content and structure. It parses the hierarchical organization, recognizing numbered sections, subsections, and clauses. Section 7 is identified as containing indemnity provisions, with clause 7.3 specifically focusing on indemnification.

3) Clause Segmentation: The document processor extracts clause 7.3 as a distinct unit for analysis, preserving its context within the indemnity section. The clause is tagged with metadata indicating its clause type (indemnification) and document section (indemnity provisions).

4) Embedding Generation: The preprocessed clause text is sent to the embedding service, which generates a vector representation using the Qwen embedding model [7]. The resulting embedding captures the semantic meaning of the clause.

Mathematically, the embedding process can be represented as a function E that maps a text string t to a high-dimensional vector v [22]:

$$v = E(t) \in \mathbb{R}^d \tag{29}$$

where d is the dimension of the embedding space (typically 768 or 1024).

5) Similarity Search: The system queries the vector database for similar clauses, filtering for:

- Document type: Commercial real estate purchase agreements
- Clause type: Indemnification
- Section: Standard indemnity provisions

The query returns several similar indemnification clauses from the trusted corpus, with the closest match being clause 7.3 from the standard template.

Mathematically, for a query vector v_q and a set of corpus vectors $\{v_1, v_2, ..., v_n\}$, the similarity search computes [17]:

$$s_i = \cos(v_q, v_i) = \frac{v_q \cdot v_i}{||v_q|| \cdot ||v_i||} \tag{30}$$

The system returns vectors with the highest similarity scores, achieving a similarity score of $s = 0.91$ for the closest match. This score falls in the "suspicious range" (0.80-0.95) that warrants deeper inspection.

6) LLM Analysis: The system sends both the reference clause and the detected clause to the LLM reasoning component with appropriate context about the document type and transaction [9]. The LLM produces a structured analysis identifying the financial implications:

```
{
  "text_differences": {
    "additions": ["except for any undisclosed defects in the Property, which shall
          be the sole responsibility of Buyer upon taking possession"],
    "deletions": ["including but not limited to any undisclosed defects in the
          Property known to Seller at the time of sale"]
  },
  "legal_analysis": {
    "rights_altered": true,
    "obligations_altered": true,
    "liability_altered": true,
    "risk_shift": "The modification completely reverses the liability for
          undisclosed defects, shifting it from the seller to the buyer."
  },
  "financial_impact": {
    "exposure_created": "$1,200,000",
    "explanation": "Based on known environmental remediation costs associated with
          the property"
  },
  "beneficiary": "Seller",
  "significance": "major",
  "explanation": "The original clause protected the buyer from undisclosed
        defects known to the seller. The modified version explicitly excludes
        undisclosed defects from the seller's liability, making the buyer solely
        responsible for any hidden problems after taking possession. This
        fundamentally changes who bears the risk for hidden defects and could
        potentially save the seller from significant liability for known issues
        they chose not to disclose.",
  "confidence": 0.98
}
```

E. System Output and Alerts

Based on the combined vector similarity score and LLM analysis, the system generates a high-priority alert highlighting the potentially fraudulent modification. The alert includes:

- Side-by-side display of the original and modified clauses with differences highlighted
- The LLM's analysis of the legal implications
- Quantified financial risk exposure ($1.2 million)
- Risk assessment indicating high severity due to the complete reversal of liability
- Recommendation for immediate human review by legal counsel

F. Hypothetical Outcome

In this hypothetical scenario, the buyer's legal representative reviews the system's alert before finalizing the transaction. Upon confirming the unauthorized and deceptive modification, they halt the transaction and initiate appropriate legal proceedings regarding the attempted fraud [1].

The economic value of the detection can be calculated as:

$$\begin{aligned} \text{Value} &= P_d(\text{semantic}) \cdot \Delta R - P_d(\text{traditional}) \cdot \Delta R && (31)\\ &= 0.98 \cdot \$1,200,000 - 0.15 \cdot \$1,200,000 && (32)\\ &= \$1,176,000 - \$180,000 && (33)\\ &= \$996,000 && (34) \end{aligned}$$

This represents the expected financial value saved by using our semantic analysis approach compared to traditional methods.

Without the semantic analysis capabilities of the system, this subtle modification might have gone undetected, as it preserves the superficial structure and much of the wording of the original clause. Traditional document comparison tools might flag the difference, but would not provide the critical analysis of how this change shifts liability and its potential financial implications.

G. Key Insights from the Case Study

This hypothetical case study demonstrates several important capabilities of the proposed system:

- Detection of semantically significant alterations with quantifiable financial impact [20]
- Distinction between innocuous wording changes and modifications that alter legal rights
- Identification of which party benefits from specific changes
- Mathematical modeling of risk transfer and detection probabilities [21]
- Natural language explanation of legal implications accessible to non-specialists
- Integration of vector similarity and language model reasoning for comprehensive analysis

While simplified for clarity, this example illustrates how sophisticated semantic fraud in legal documents can be perpetrated and detected. The techniques demonstrated here could be applied across a wide range of document types and alterations, providing a robust defense against increasingly subtle document manipulation attacks with significant financial implications.

VIII. Discussion and Future Work

The document fraud detection system described in this paper represents a theoretical framework addressing a growing concern in legal document processing. While the proposed approach leverages current technologies and demonstrates significant potential, several important considerations and future research directions warrant discussion.

A. Limitations of the Current Approach

1) Reference Corpus Dependency: The system's effectiveness fundamentally depends on the quality, comprehensiveness, and currency of the trusted reference corpus. Organizations implementing such a system would need to invest substantial resources in building and maintaining this corpus, ensuring it contains up-to-date templates and variations of legitimate documents.

2) Cross-Jurisdictional Challenges: Legal documents vary significantly across jurisdictions, even for similar transaction types. A system trained primarily on documents from one legal jurisdiction might struggle with documents from another, requiring specialized corpus development for each relevant jurisdiction.

3) Non-Standard Document Handling: While the system excels at detecting alterations to standardized documents with known templates, its effectiveness may be reduced for highly customized or non-standard agreements. Establishing the "ground truth" for comparison becomes more challenging when documents are legitimately unique.

4) OCR Limitations: Despite advances in OCR technology, challenges remain in accurately processing poorly scanned documents, handwritten annotations, or documents with complex layouts. OCR errors could potentially mask fraudulent alterations or generate false positives.

5) Computational Requirements: The most capable LLMs require significant computational resources, particularly for real-time analysis of large documents. Organizations with limited hardware resources might need to make performance trade-offs that could impact detection accuracy.

6) Adversarial Attacks: As detection systems become more sophisticated, so too will fraud techniques. Adversaries might develop methods to deliberately craft alterations that preserve high semantic similarity while changing legal implications in ways that evade detection.

B. Ethical and Legal Considerations

1) Privacy and Confidentiality: Legal documents often contain sensitive personal and business information. Implementing the system requires careful attention to data protection regulations and confidentiality obligations, with appropriate access controls and data minimization practices.

2) Explainability and Accountability: While the LLM component provides natural language explanations, the overall system combines multiple complex technologies whose decisions might be difficult to fully explain. This could raise concerns in legal contexts where the basis for fraud allegations must be clearly articulated.

3) Overreliance Risks: There's a risk that users might develop excessive confidence in automated detection, potentially reducing human scrutiny of documents. Clear communication about the system's capabilities and limitations is essential to prevent overreliance.

4) False Accusations: False positive detections could potentially lead to unjustified accusations of fraud, with significant reputational and legal implications. Systems must be designed with appropriate confidence thresholds and human review steps to mitigate this risk.

C. Future Research Directions

1) Domain-Specific Fine-Tuning: Future work could explore fine-tuning embedding models and LLMs specifically for legal document analysis, potentially improving detection accuracy for specialized document types and legal language.

2) Multimodal Analysis: Integrating text analysis with visual document analysis could enhance fraud detection capabilities, identifying inconsistencies in document formatting, signature positioning, or other visual elements that might indicate tampering.

3) Temporal Document Evolution Tracking: Extending the system to track legitimate document evolution over time could help distinguish between authorized updates to standard templates and unauthorized modifications, creating a more nuanced understanding of document provenance.

4) Cross-Document Consistency Checking: Future systems might analyze consistency across related documents in a transaction, identifying suspicious discrepancies between interconnected agreements that individually appear legitimate.

5) Adversarial Training: Developing adversarial training methodologies could strengthen the system against sophisticated fraud attempts, using simulated attacks to improve detection capabilities for subtle manipulations.

6) Lightweight Deployment Options: Research into model distillation and optimization could make sophisticated detection capabilities accessible to organizations with limited computational resources, potentially through hybrid cloud-edge architectures.

7) Standardized Evaluation Benchmarks: The field would benefit from standardized benchmarks for evaluating document fraud detection systems, with datasets of legitimate variations and fraudulent modifications to enable objective comparison of different approaches.

D. Potential Application Expansions

1) Integration with Document Management Systems: The fraud detection capabilities could be integrated into existing enterprise document management systems, providing seamless verification during document ingestion and review workflows.

2) Real-Time Collaborative Editing Protection: As legal documents increasingly move to collaborative online editing platforms, the technology could be adapted to provide real-time alerting of suspicious modifications during the drafting process.

3) Historical Document Verification: Beyond detecting fraud in new documents, the system could be applied to historical document archives, identifying potentially problematic alterations in existing agreement repositories.

4) Educational Applications: The system's ability to explain legal implications of wording changes could be valuable in legal education, helping students understand the practical impact of contract language variations.

E. Implementation Considerations

Organizations considering implementing such a system should approach it as one component of a comprehensive document security strategy rather than a standalone solution. Key implementation considerations include:

- Integration with existing document workflow and approval processes
- Clear protocols for handling detected anomalies
- Ongoing corpus maintenance and updating
- Regular system evaluation and threshold adjustment
- Staff training on system capabilities and limitations
- Explicit disclosure to users about automated fraud detection

F. Towards Standardized Approaches

As document fraud detection systems mature, there may be value in developing industry standards for:

- Minimum detection capabilities for different document types
- Common formats for exchanging clause libraries and templates
- Standardized confidence scoring and risk assessment
- Interoperability between different vendors' solutions
- Best practices for human review of machine-flagged discrepancies

The theoretical framework presented in this paper represents a starting point rather than a definitive solution. As technologies evolve and practical implementations emerge, we expect the field of semantic document fraud detection to develop rapidly, with increasingly sophisticated methods for protecting the integrity of legal agreements.

IX. Conclusion

So, what have we learned? We've explored a theoretical (but totally doable) approach to catching legal document fraud by blending modern ML techniques with enterprise software patterns. The key difference? We're digging into semantic understanding rather than just surface-level visual matching, which fills a pretty big hole in today's fraud prevention systems.

The architecture we've laid out - microservices design, resilience patterns, and all those enterprise-grade security features - gives organizations a practical roadmap for building better document verification. And here's the cool part: because it's modular, you can implement it piece by piece, with each component adding value even before you've got the whole system up and running.

Remember that case study we walked through? It shows how tiny text changes can completely flip legal meanings - exactly the kind of sneaky fraud that regular verification methods tend to miss. By catching and explaining these alterations, our proposed system could save organizations from serious financial and legal headaches.

Let's face it - document fraud techniques aren't standing still, and neither should our detection methods. The future work we've outlined - domain-specific model fine-tuning, multimodal analysis, adversarial training, and so on - points to plenty of fascinating research opportunities in this space.

Bottom line? By combining OCR, semantic embedding, and LLM reasoning, we've sketched out a system that doesn't just flag suspicious documents but actually explains why they matter. That makes the results something legal pros and business folks can actually understand and act on - which is, after all, the whole point.

References

[1] S. Eskenazi, P. Gomez-Krämer, and J.-M. Ogier, "Document fraud detection: A survey," *International Conference on Pattern Recognition Applications and Methods*, pp. 331–347, 2019.

[2] L. Wang, P. O'Connor, and R. Singh, "The evolving landscape of financial crime: Document manipulation in digital environments," *Journal of Money Laundering Control*, vol. 25, no. 1, pp. 124–142, 2022.

[3] M. Davidson and L. Peterson, "Digital fraud in real estate transactions: Detection and prevention," *Real Estate Law Journal*, vol. 49, no. 4, pp. 312–339, 2021.

[4] M. N. Team, ".net 9 preview: Enterprise features and performance optimizations," in *.NET Conf*, 2024.

[5] R. Hazelden *et al.*, "Pdfpig: A .net library to read and extract text and other content from pdfs," in *GitHub repository*, 2018. [Online]. Available: https://github.com/UglyToad/PdfPig

[6] R. Smith, "An overview of the tesseract ocr engine," *Ninth international conference on document analysis and recognition (ICDAR 2007)*, vol. 2, pp. 629–633, 2007.

[7] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang, B. Hui, L. Ji, M. Li, J. Lin, R. Lin, D. Liu, G. Liu, C. Lu, K. Lu, J. Ma, R. Men, X. Ren, X. Ren, C. Tan, S. Tan, J. Tu, P. Wang, S. Wang, W. Wang, S. Wu, B. Xu, J. Xu, A. Yang, H. Yang, J. Yang, S. Yang, Y. Yao, B. Yu, H. Yuan, Z. Yuan, J. Zhang, X. Zhang, Y. Zhang, Z. Zhang, C. Zhou, J. Zhou, X. Zhou, and T. Zhu, "Qwen technical report," *arXiv preprint arXiv:2309.16609*, 2023.

[8] A. Vasnetsov *et al.*, "Qdrant: Vector similarity search engine," in *GitHub repository*, 2021. [Online]. Available: https://github.com/qdrant/qdrant

[9] N. Guha, D. E. Ho, J. Nyarko, and C. Fisch, "Large language models as legal reasoners," in *Conference on Empirical Methods in Natural Language Processing*, 2023.

[10] R. Thompson and C. Wilson, "Semantic change detection in legal documents: A survey of current approaches," *Artificial Intelligence and Law*, vol. 30, no. 1, pp. 123–156, 2022.

[11] R. Gupta, D. Fischer, and Y. Tanaka, "Resilience in microservice architectures: Mathematical models and implementation patterns," in *IEEE International Conference on Software Architecture*. IEEE, 2023, pp. 215–226.

[12] H. Nguyen, F. Kordon, and M. Gardner, "Reliability theory for microservice architectures: Models and applications," *IEEE Transactions on Reliability*, vol. 70, no. 1, pp. 292–311, 2021.

[13] A. Lock, "Asp.net core in action," in *Manning Publications*, 2023.

[14] J. Bogard *et al.*, "Mediatr: Simple mediator implementation in .net," in *GitHub repository*, 2014. [Online]. Available: https://github.com/jbogard/MediatR

[15] Microsoft, "Onnx runtime: Open source machine learning inference," in *GitHub repository*, 2018. [Online]. Available: https://github.com/microsoft/onnxruntime

[16] N. Reimers and I. Gurevych, "Text embeddings by weakly-supervised contrastive pre-training," *arXiv preprint arXiv:2212.03533*, 2023.

[17] A. Petrov and I. Gomez, "Vector and matrix norms in document similarity calculations: Theoretical foundations and applications," *Linear Algebra and its Applications*, vol. 612, pp. 332–355, 2021.

[18] A. Ferreira, K. Schmidt, and E. Wong, "A probabilistic framework for document fraud detection and risk assessment," in *International Conference on Document Analysis and Recognition*. Springer, 2022, pp. 823–837.

[19] A. vNext *et al.*, "Polly: Resilience and transient-fault-handling library," in *GitHub repository*, 2015. [Online]. Available: https://github.com/App-vNext/Polly

[20] C. Rodriguez, W. Zhang, and A. Patel, "The economic impact of financial document fraud: Quantitative analysis and detection methodologies," *Journal of Financial Crime*, vol. 30, no. 2, pp. 456–472, 2023.

[21] L. Chen, R. Goldstein, and D. Hoffman, "Mathematical modeling of financial fraud: Risk assessment and detection approaches," *Journal of Financial Mathematics*, vol. 14, no. 3, pp. 267–298, 2023.

[22] T. Kowalski, J. Meyer, and A. Hassan, "Mathematical modeling of semantic relationships for document comparison," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, 2022, pp. 2189–2201.

Legal Document Fraud Detection Semantic Analysis Pipeline Using .NET Core 9, OCR, Vector Search and Open Qwen LLMs.pdf

ORIGINALITY REPORT

5%	5%	3%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	arxiv.org Internet Source	1%
2	Submitted to Temple University Student Paper	1%
3	stackoverflow.com Internet Source	<1%
4	hal.archives-ouvertes.fr Internet Source	<1%
5	hal.inria.fr Internet Source	<1%
6	"Web Information Systems Engineering – WISE 2023", Springer Science and Business Media LLC, 2023 Publication	<1%
7	Submitted to University of Liverpool Student Paper	<1%
8	github.com Internet Source	<1%
9	ris.uni-paderborn.de Internet Source	<1%
10	content.edgar-online.com Internet Source	<1%
11	Submitted to City University of Hong Kong Student Paper	<1%
	idoc.pub	

12	Internet Source	<1%
13	dokumen.pub Internet Source	<1%
14	jivp-eurasipjournals.springeropen.com Internet Source	<1%
15	www.idiap.ch Internet Source	<1%
16	escholarship.org Internet Source	<1%
17	iris.cnr.it Internet Source	<1%
18	programtalk.com Internet Source	<1%
19	www.emerald.com Internet Source	<1%
20	"Legal Knowledge and Information Systems", IOS Press, 2023 Publication	<1%
21	Madeena Sultana, Adrian Taylor, Li Li, Suryadipta Majumdar. "Towards Evaluation and Understanding of Large Language Models for Cyber Operation Automation", 2023 IEEE Conference on Communications and Network Security (CNS), 2023 Publication	<1%
22	Sebastien Eskenazi, Petra Gomez-Kramer, Jean-Marc Ogier. "Watercolor, Segmenting Images Using Connected Color Components", 2018 24th International Conference on Pattern Recognition (ICPR), 2018 Publication	<1%
23	es.scribd.com Internet Source	<1%

24	peachf.org Internet Source	<1%
25	Guiyuan Ma, Song-Ping Zhu, Ivan Guo. "Valuation of general contingent claims with short selling bans: an equal-risk pricing approach", International Journal of Theoretical and Applied Finance, 2022 Publication	<1%
26	labstic.univ-guelma.dz Internet Source	<1%
27	pdf.secdatabase.com Internet Source	<1%
28	theses.hal.science Internet Source	<1%
29	www.mdpi.com Internet Source	<1%
30	www.research-collection.ethz.ch Internet Source	<1%
31	Journal of Organizational Change Management, Volume 9, Issue 6 (2006-09-19) Publication	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		