

Semantic Change Detection and Persona-Aware Revision of Educational Content Using Bidirectional Vector Retrieval and LLMs

Manas Tarare¹, Damodhar Meshram²

^{1,2}Morningmind technolab private limited, LRN Technologies, Nagpur, Maharashtra, India

Abstract:

Content in educational materials can quickly become outdated due to changes in reliable sources, necessitating ongoing manual revisions that risk introducing errors. This paper introduces a system for detecting and classifying semantic changes in educational content using structured document parsing, semantic embeddings, bidirectional vector retrieval, and LLM-based generation. The approach preserves structural integrity while identifying links to verified content and categorizing changes as Edit, Addition, Deletion, or Context Change. Revisions are generated based on a persona profile to maintain writing style, followed by human review for final approval. The prototype demonstrates an efficient end-to-end process for automatic content change detection and controlled revisions with human oversight.

Keywords: Semantic change detection, bidirectional vector retrieval, semantic embeddings, large language models, persona-aware text generation, educational content, human-in-the-loop.

1. INTRODUCTION

Educational content, including textbooks, course material, and institutional training resources, requires periodic revision as policies, regulations, scientific findings, and reference sources evolve. Conventional maintenance practices commonly depend on manual comparison, keyword matching, or surface-level text search. These approaches are limited when the wording remains similar but the underlying meaning changes, or when new source information is absent from the existing document.

Recent advances in semantic embeddings and large language models (LLMs) provide an opportunity to analyze content at the level of meaning rather than exact wording. This paper presents the AutoDetect Content Change System, an automated pipeline for detecting and revising outdated or incomplete educational content with respect to a corresponding Source-of-Truth (SoT). The system combines structured document parsing, semantic retrieval, LLM-based verification, persona-aware revision generation, and human review.

The proposed system identifies four major types of content change: Edit, where existing factual information has changed; Addition, where new source information is missing from the document; Deletion, where obsolete or unsupported content should be removed; and Context Change, where changed circumstances require the content to be reframed. These categories support targeted revision while avoiding unnecessary rewriting of unaffected material.

2. PROBLEM IDENTIFICATION

Educational content such as textbooks, course modules, and institutional training material frequently becomes outdated when authoritative sources are revised. Existing maintenance practices rely heavily on manual comparison, keyword search, and reviewer judgment. Although these methods can identify obvious textual differences, they are often ineffective when the wording remains similar but the underlying meaning changes. As a result, outdated or incomplete information may remain in learning material, reducing reliability and increasing the risk of misinformation.

The main problem addressed in this work is the automatic detection and controlled revision of educational content when its corresponding source of truth changes. The system must identify whether a change requires editing existing content, adding missing information, deleting obsolete material, or reframing content due to a change in context. This classification is necessary because different types of changes require different revision actions.

Several challenges make this problem difficult. First, semantic changes may not be visible through direct keyword matching. For example, two passages may use similar terms while referring to different policies, conditions, or target groups. Second, a one-way comparison between the existing document and the source of truth is insufficient, since it may fail to detect both missing information and unsupported content. Third, educational documents often contain structured elements such as headings, lists, and tables, which must be preserved during revision. Treating the document as plain unstructured text can damage its organization and readability.

Another important challenge is maintaining the writing style of the original document. Revisions should not only be factually correct but also consistent with the tone, vocabulary, and structure of the source material. Fully automatic rewriting can introduce factual errors, stylistic inconsistency, or biased modifications. Therefore, human review is required before changes are applied to the final document.

The goal of the proposed system is to reduce the manual effort required to detect content changes while preserving reviewer control over final revisions. The system should ingest an educational document and a source-of-truth document, detect and classify factual changes, generate revision suggestions, and present each suggestion with sufficient context for human approval. A successful system should improve review efficiency, reduce missed updates, and maintain the structural and stylistic integrity of the original document.

3. RELATED WORK

Document change detection has traditionally been studied in the context of stylistic variation, discourse shifts, and text similarity rather than factual content revision. Alvi et al. [1] used discourse markers to identify stylistic changes across text segments, while Jiang et al. [2] applied pre-trained language models with similarity recognition to detect writing-style changes within documents. These studies show that similarity-based methods can identify stylistic discontinuities, but they do not directly address whether the factual meaning of the content has changed. This distinction is important for educational content maintenance, where the system must determine whether information should be edited, added, deleted, or reframed due to contextual change.

Semantic retrieval methods provide a stronger foundation for identifying meaning-level relationships between document passages. Yang et al. [3] demonstrated that multilingual sentence encoders can represent text in a shared vector space, enabling effective semantic similarity search across languages and domains. Similar retrieval principles are used in modern document-processing systems, where text chunks

are embedded and compared based on meaning rather than exact word overlap. Boddupally [4] further discussed the role of semantic retrieval in enterprise-scale knowledge systems, emphasizing the importance of combining embeddings with structured metadata. This is relevant to educational document revision because document structure, section context, and chunk-level metadata must be preserved during comparison.

Although semantic retrieval can identify potentially related passages, similarity scores alone are not sufficient for determining the type of change required. Large language models have shown strong potential for structured text analysis and classification. Wan et al. [5] demonstrated that LLMs can extract and categorize information from large volumes of unstructured text into meaningful labels. Such capabilities are useful for classifying retrieved document pairs into change categories such as factual edits, missing information, obsolete content, or contextual changes. However, existing work mainly focuses on detection or classification and gives limited attention to generating replacement content that remains consistent with the original author's tone, vocabulary, and document structure.

Human oversight is also an important requirement in automated content revision systems. Kumar et al. [6] reviewed human-in-the-loop learning systems and highlighted the need for human validation when model confidence is uncertain or when incorrect automated decisions may have significant consequences. This is especially relevant in educational content, where factual errors or misleading revisions can affect learners. Therefore, automated systems should support reviewers by identifying likely changes and suggesting revisions, while leaving final approval to human judgment.

However, limited work has integrated these components into a single pipeline for fact-level educational content change detection and persona-consistent revision generation. The proposed AutoDetect system addresses this gap by combining structured document parsing, bidirectional semantic retrieval, LLM-based change classification, persona-aware revision generation, and human review.

Figure 1 represents a simple architecture flow of the system that we have designed

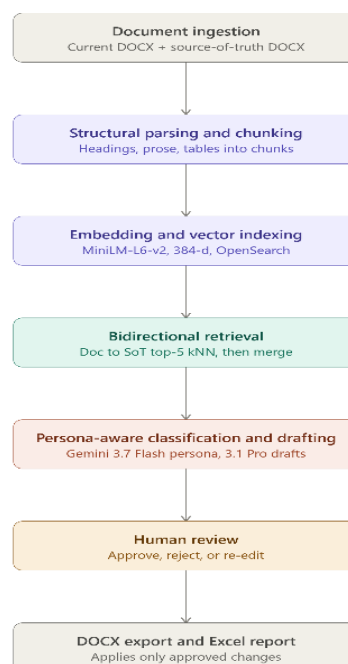


Fig. 1. Architecture workflow

4. METHODOLOGY

A. System Overview

The AutoDetect Content Change System is designed to compare a current educational DOCX document with a related source-of-truth DOCX document. Its purpose is to identify content that may need to be updated, added, removed, or rewritten because the source-of-truth document contains different information. The system works in two main phases. First, both documents are uploaded, read, divided into meaningful parts, and prepared for comparison. Second, the system compares the parts from both documents, identifies possible changes, prepares revision suggestions, and sends them to a human reviewer. The main inputs are a current educational document and a source-of-truth document. The main outputs are possible content changes, suggested revisions, reviewer decisions, an Excel change report, and an updated DOCX document.

B. Document Ingestion

The system accepts two uploaded files. One file is assigned the role of the current document, and the other file is assigned the role of the source of truth. Each uploaded file is stored with its filename, file type, document role, file size, and content. Although the upload step can recognize both .doc and .docx extensions, the working parsing process supports DOCX files only. Therefore, the prototype should be used with DOCX documents. The document role is preserved throughout the process. This allows the system to distinguish between the current document and the source-of-truth document during comparison.

C. Structural Parsing

The system reads the DOCX document in its original order and identifies different types of content, including headings, short subheadings, normal paragraphs, bullet points, callout-style text, and tables. For each paragraph, the system records useful details such as the text, style name, font name, font size, bold formatting, shading, borders, and list information. Headings are identified through heading styles and font-size patterns. Short paragraphs that are not headings, lists, or callouts are treated as subheadings. Other paragraphs are treated as normal prose. Lists are identified through Word numbering settings, list styles, and visible bullet symbols. Callout-like text is identified through borders, shading, special styles, or starting words such as Note, Warning, Important, Tip, Caution, and Disclaimer. Tables are read as rows and cells while keeping their original location in the document. The output of this stage is a structured representation of the document. It includes the text, content type, location in the document, parent heading, section path, style details, and document role.

D. Structured Chunking

After parsing, the system divides the documents into smaller meaningful units called chunks. Each chunk is used as one comparison unit. Headings, subheadings, bullet points, callouts, and tables are kept as separate chunks. Tables are changed into readable text while keeping their row and column structure. Normal paragraphs are kept as one chunk when they contain 200 words or max 250 words. Paragraphs longer than 250 words are split at sentence boundaries. The system aims to create chunks of about ~200 words (max 250 words) so that long paragraphs can be compared more easily without cutting sentences in the middle. Each chunk receives a stable identifier based on the filename, document role, position, and text. This means that the same document produces the same chunk identifier when the notebook is run again without changes.

E. Semantic Embedding Generation

The system converts each chunk into a numeric representation that allows similar meanings to be compared even when the wording is different. The notebook uses the sentence-transformers/all-MiniLM-

L6-v2 model. This model creates a 384-dimensional representation for each chunk. The representations are normalized so that the comparison process can measure how closely two chunks are related in meaning. Each prepared chunk record contains the chunk ID, filename, source role, content type, raw text, model name, embedding dimension, numeric representation, and structural metadata. The notebook saves these records in JSON and JSONL formats for inspection and indexing.

F. OpenSearch-Based Vector Storage

The system stores prepared chunk records in OpenSearch so that chunks from the two documents can be compared efficiently. The main index is named `doc_chunks`. It stores chunk ID, filename, source role, content type, raw text, embedding information, and metadata. The metadata includes information such as chunk position, block index, paragraph or table index, parent heading, section path, formatting details, and word count. The system uses a separate index named `persona_profiles` to store the writing-style profile of the current document. This keeps the document chunks and writing-style information separate. After indexing, the notebook checks the total number of stored chunks and separately checks the number of chunks from the current document and the source-of-truth document.

G. Persona Profile Extraction

The system creates a writing-style profile from the current document so that revision suggestions can match the original writing style. The notebook selects representative chunks from the beginning, middle, and end of the current document. It also tries to select different content types, such as prose, headings, bullets, callouts, and tables. The selected samples are sent to the language model. The model creates a profile describing the writing style, including tone, level of formality, vocabulary level, sentence style, use of active or passive voice, formatting habits, and content-type-specific patterns. The completed writing-style profile is stored in the `persona_profiles` OpenSearch index and is later used when drafting revision suggestions.

H. Bidirectional Change Detection

The system compares the current document and source-of-truth document in both directions. This is important because a one-way comparison may find changed existing content but miss new information that should be added. In the first comparison direction, each chunk from the current document is compared with the closest chunks from the source-of-truth document.

In the second direction, each source-of-truth chunk is compared with the closest chunks from the current document. The system retrieves up to five close matches for each chunk. A match with a score of at least 0.70 is treated as a high-confidence relationship between the two chunks. High-confidence matches are treated as related candidate pairs and are sent for revision generation. In the current batch workflow, matched candidate pairs are assigned the default change type of Edit unless another type has already been provided. Chunks that do not receive a high-confidence match are stored as unknown candidates. An unmatched source-of-truth chunk may represent missing information that should be added. An unmatched current-document chunk may represent content that should be deleted or reviewed. For each unknown candidate, the system also keeps the three closest low-confidence references from the other document. These references help the language model decide whether the candidate is a real change or should be ignored.

I. Candidate Pair Assembly

Candidate pairs from both comparison directions are merged using the current-document chunk ID and source-of-truth chunk ID. If the same pair appears in both comparison directions, both scores are retained. The system calculates a combined score by averaging the available comparison scores. Candidate pairs

with a combined score below 0.65 are removed. The remaining pairs are sorted from the highest score to the lowest score. Unmatched source-of-truth chunks and unmatched current-document chunks are stored as unknown candidates rather than confirmed additions or deletions.

The language model later decides whether an unknown candidate represents Addition, Deletion, or Ignore. The notebook creates an Excel file named `changes_report.xlsx`. This report includes Edit and Context Change candidate pairs, unknown candidates, chunk IDs, current-document text, source-of-truth text, comparison scores, and comparison directions.

Table 1 illustrates one candidate pair row from `changes_report.xlsx`, showing the key fields captured for each comparison — chunk IDs, document/SoT text, combined score, and comparison direction.

Table 1. Example from `changes_report.xlsx` (one candidate pair row)

Field	Example Value
<code>change_category</code>	Edit/Context Change candidate
<code>content_type</code>	prose
<code>document_chunk_id</code>	01b45459... 26632c63
<code>document_text</code>	“Reinforcement learning takes its inspiration from animal psychology. An agent takes actions in an environment, receives rewards or penalties...”
<code>sot_chunk_id</code>	454725dc... 4ad0e99c77
<code>sot_text</code>	“This example is outdated. DeepMind’s AlphaZero proved that a single general-purpose algorithm can master multiple games—chess, shogi, and Go—from scratch...”
<code>combined_score</code>	0.8086
<code>directions</code>	Doc→SoT; SoT→Doc
<code>best_score</code>	—

J. Change Classification and Drafting

The system prepares reviewer-ready suggestions from the candidate pairs and unknown candidates. For matched candidate pairs, the system compares the current-document text with the source-of-truth text and prepares a revision suggestion. In the current batch workflow, these matched pairs are assigned Edit as the default change type. The notebook also contains support for Context Change as an available change category, but the active batch workflow does not automatically classify matched pairs as Context Change. For unknown source-of-truth candidates, the language model decides whether the information is genuinely missing from the current document. The result is classified as Addition or Ignore. For unknown current-document candidates, the language model decides whether the content is unsupported by the source of truth. The result is classified as Deletion or Ignore. For Addition cases, the system prepares new text. For Edit and Context Change cases, it prepares replacement text. For Deletion and Ignore cases, no replacement text is prepared.

K. Persona-Aware Generation

The writing-style profile is used when generating revision suggestions. The system instructs the language model to keep the original document’s tone, vocabulary, sentence style, and general writing approach. For Edit and Context Change suggestions, the system aims to change only the content needed for the update. For Addition suggestions, it aims to create new text that fits the style of the current document. The

generated result includes the change type, original text, source-of-truth text where available, proposed text where applicable, reason for the change, and confidence information returned by the language model.

L. Human Approval Workflow

All generated suggestions are reviewed by a human before they are applied to the document. The review interface shows the change type, source type, number of re-edits, original text, source-of-truth text, proposed text, and reason for the suggested change. The reviewer can move through generated suggestions using Previous and Next controls.

The reviewer can choose Approve, Reject, or Re-edit. Approval marks the suggestion for application. Rejection keeps the original document content unchanged. Re-edit allows the reviewer to provide feedback and request a revised suggestion. The system does not apply changes automatically. Human approval is required before any suggested revision is included in the updated document.

M. Final DOCX Export

The prototype includes a DOCX export step that creates an updated copy of the original document. Approved Edit and Context Change suggestions are applied by replacing the related paragraph text. Approved Addition suggestions are added at the end of the document because the current system does not yet identify a precise insertion position for new content. The export process has some limitations. When a paragraph is replaced, only the first text run's formatting is retained, so mixed formatting inside a paragraph may not be fully preserved. Also, deletion suggestions from the active unknown-candidate workflow may not always be applied because they do not always retain the original document block location. Review progress is retained in memory during the notebook session and is also saved in an OpenSearch checkpoint index. This allows the review queue, approvals, rejections, and review history to be restored later.

5. SYSTEM IMPLEMENTATION

The AutoDetect Content Change System was implemented as a Python notebook in Google Colab. The implementation follows a sequential pipeline in which each stage produces data that is consumed by later stages. The system accepts two Word documents: a current educational document and a source-of-truth document. It parses both documents, converts their content into structured chunks, generates semantic embeddings, performs bidirectional similarity retrieval, creates change candidates, generates revision suggestions with Gemini, supports human review, and exports an updated DOCX file.

The implementation uses python-docx for DOCX parsing and export, sentence-transformers for semantic embeddings, opensearch-py for vector storage and similarity search, openai for the Gemini API connection, openpyxl for report generation, ipywidgets for the review interface, and matplotlib and seaborn for evaluation charts.

A. Document Ingestion and File Management

The notebook uses the Google Colab upload interface to collect two input files. The first file is assigned the role document, representing the current educational document. The second file is assigned the role sot, representing the source-of-truth document. Each uploaded file is stored in an in-memory dictionary named `ingested_documents`.

The stored record includes the filename, detected file type, document role, binary file content, and file size. The upload function recognizes both .doc and .docx extensions. However, the actual parsing pipeline uses python-docx, which supports DOCX files. Therefore, the working implementation processes DOCX documents only. Non-DOCX files are skipped during parsing.

B. DOCX Structural Parsing

The parser reads the DOCX file using python-docx and iterates through the document body in its original order. It processes paragraph blocks and table blocks separately while retaining their document positions through block indexes. For every paragraph, the parser extracts the raw text, style name, font name, font size, bold setting, shading, border information, numbering properties, and list metadata. These attributes are used to classify paragraphs into headings, subheadings, prose, bullets, and callouts. A paragraph is identified as a heading when it uses a Heading 1 or Title style, or when its font size is sufficiently larger than the dominant body-text font size. Short paragraphs that are not headings, list items, or callouts are classified as subheadings. Paragraphs containing more than ten words are generally classified as prose. The parser detects list items through Word numbering settings, list-related styles, or visible text markers such as bullets, hyphens, numbers, and letters. Callout-style content is detected through paragraph shading, borders, specific styles, or leading keywords such as Note, Warning, Important, Tip, Caution, and Disclaimer. Tables are extracted as row-cell arrays and kept in their original sequence. Text contained in Word text boxes is also detected and treated as callout content. The parsing stage produces structured document elements, tables, callouts, document roles, and file-type information.

C. Structured Chunk Construction

The parsed output is transformed into chunks. A chunk is the smallest unit used for embedding, retrieval, comparison, classification, and review. Headings, subheadings, bullet points, callouts, and tables are stored as independent chunks. Tables are converted into readable row-based text by joining non-empty cell values with separators. This preserves the table content while making it suitable for semantic comparison. Prose paragraphs are retained as one chunk when they contain 200 words. If a prose paragraph exceeds 200 words, it is split at sentence boundaries using a regular-expression sentence splitter. The target chunk size is approximately 200 words. This avoids cutting sentences in the middle and reduces the chance that a large paragraph hides a smaller factual change. Every chunk includes a deterministic chunk_id, filename, source role, content type, raw text, chunk position, and metadata. The chunk ID is generated using a SHA-256 hash of the filename, source role, chunk position, and raw text. This provides stable IDs when the notebook is rerun with unchanged input documents.

The metadata stores structural information such as block index, paragraph index, table index, parent heading, section path, list properties, style name, font details, bold state, word count, and long-paragraph split information.

D. Embedding Generation and Vector Storage

The notebook uses sentence-transformers/all-MiniLM-L6-v2 to generate semantic embeddings for all non-empty chunks. The model produces 384-dimensional vectors. Embeddings are generated in batches of 32 and are normalized before storage. Normalization ensures that similarity retrieval is based on consistent cosine-style comparison. Each indexed record includes the chunk ID, filename, source role, content type, raw text, embedding model name, embedding dimension, embedding vector, and metadata. The embedding records are saved locally as chunk_embeddings_opensearch.json and chunk_embeddings_opensearch.jsonl. The JSON file is intended for inspection, while JSONL supports bulk ingestion workflows. The main OpenSearch index is named doc_chunks. Its mapping contains keyword fields for chunk identifiers, filename, source role, and content type; a text field for raw content; an object field for metadata; and a knn_vector field for embeddings. The vector index uses HNSW nearest-neighbor search with cosine similarity. The notebook uses bulk indexing to insert chunk records into OpenSearch. After indexing, the index is refreshed and counted. The implementation separately reports

the number of chunks belonging to the document and sot roles. The notebook also creates a separate `persona_profiles` index. This index stores the writing-style profile created from the current document. A third index, `review_checkpoints`, stores review progress so that a review session can be restored after interruption.

E. Persona Profile Generation

The system builds a writing-style profile from representative chunks of the current document. It first divides the current-document chunks into early, middle, and final portions. It then selects up to five samples while trying to include varied content types. The selected samples are sent to Gemini with a structured prompt. The prompt requests information about tone, formality, vocabulary complexity, sentence structure, paragraph style, active or passive voice preference, formatting conventions, content-type-specific patterns, and other writing characteristics. Persona extraction uses `gemini-3.7-flash`. Gemini is accessed through the OpenAI-compatible Google Generative Language API endpoint. The API key is read from the `GEMINI_API_KEY` environment variable or entered manually through `getpass`. The persona response is expected in JSON format. Before indexing, nested profile fields are converted into text where necessary so that they can be stored safely in OpenSearch text fields. The profile is stored under the current `DOCUMENT_ID`.

F. Bidirectional Retrieval and Candidate Detection

The main comparison stage uses bidirectional retrieval. The first pass searches from the current document to the source-of-truth document. The second pass searches from the source-of-truth document to the current document. This two-way design supports two different detection goals. Document-to-source retrieval identifies current content that may require revision or may no longer be supported. Source-to-document retrieval identifies information present in the source of truth that may be missing from the current document. The retrieval stage uses `TOP_K = 5`, meaning that each chunk can retrieve up to five nearby matches from the opposite document. A per-pass similarity threshold of 0.70 is used. Matches with a score of 0.70 or greater are stored as high-confidence candidate pairs. Headings and subheadings are excluded from the detection stage.

They remain part of the parsed and embedded document structure, but they are not sent through document-to-source retrieval, source-to-document retrieval, or unknown-pair detection. This prevents structural labels from being incorrectly flagged as additions or deletions. The retrieval process stores the current-document chunk ID and text, source-of-truth chunk ID and text, metadata from both chunks, similarity score, and retrieval direction. Chunks that do not receive a high-confidence match are stored as `unknown_pairs`. An unmatched current-document chunk is a potential Deletion candidate. An unmatched source-of-truth chunk is a potential Addition candidate. For each unknown pair, the notebook retrieves the three closest low-confidence references from the opposite document. These references are later provided to Gemini as supporting context.

G. Candidate Pair Merging and Reporting

High-confidence results from both retrieval directions are merged using the pair (`document_chunk_id`, `sot_chunk_id`) as the key. If a pair is found in both directions, the notebook retains both similarity scores. It calculates a `combined_score` by averaging the available directional scores. Candidate pairs are retained only when their combined score is at least 0.65. The retained pairs are sorted in descending order of combined score. These pairs are treated as strong semantic relationships between the current document and the source of truth. The notebook creates an Excel report named `changes_report.xlsx`. The report contains the change category, content type, document chunk ID, document text, source-of-truth chunk ID,

source-of-truth text, combined score, retrieval directions, and best low-confidence score. The report contains both high-confidence Edit/Context Change candidates and unknown candidates. Unknown candidates are clearly labeled as potential Addition or potential Deletion cases rather than confirmed changes.

H. Gemini-Based Change Generation

The generation and review stage configures Gemini with gemini-3.1-pro-preview. Before generation, the notebook validates the API key, model availability, access permissions, and quota status. A review queue is created from high-confidence candidate pairs and unknown pairs. Candidate pairs are assigned Edit as the default change type in the active batch workflow. Unknown pairs remain unclassified until Gemini decides their final type. The notebook sends all pending review items to Gemini in one batch request. The maximum output limit is configured as 65000 tokens. The batch prompt requires Gemini to return one JSON result for each review item. For matched candidate pairs, Gemini prepares a proposed revision. For unmatched source-of-truth chunks, Gemini chooses between Addition and Ignore. For unmatched current-document chunks, Gemini chooses between Deletion and Ignore. Each generated result includes the change type, original text, source-of-truth text, proposed text, reason for change, and confidence score. Addition results contain new text. Deletion and Ignore results contain no replacement text. The notebook validates returned change types before storing them. If Gemini returns an invalid or missing type, the result is converted to Ignore rather than being treated as a real change. If a batch response is incomplete, missing items are marked as `generation_failed` and can be retried later.

I. Human Review and Checkpointing

The human review interface utilizes ipywidgets for batch-based generation and one-by-one review. It presents original text, source-of-truth text, proposed changes, reasons for changes, change types, sources, and prior re-edits. Reviewers navigate through items using Previous and Next buttons, with options to Approve, Reject, or Re-edit. Approvals move results to `approved_changes`, while rejections go to `rejected_changes`. All reviews are logged in `review_history`, and the Re-edit function allows feedback submission. Feedback, along with the context and persona profile, is sent to Gemini for revised proposals. Review states are saved in the `review_checkpoints` OpenSearch index, which captures the review queue, changes, history, current position, and time of saving, allowing restoration using `DOCUMENT_ID` as a key.

J. DOCX Export

The notebook features an export stage that generates a revised DOCX file, incorporating only approved changes. Approved edits replace existing text in the related paragraphs, while approved additions are appended at the document's end due to the lack of defined insertion points. The export process refers to the original paragraph block index for text replacement but may not preserve mixed formatting, as it retains the formatting of the first run only. A deletion routine exists; however, deletions from unknown-pair workflows may fail to apply correctly as they lack the necessary document block index. The resulting file is named with `_updated.docx` added to the original filename.

Table 2 summarizes the main configuration parameters used in the AutoDetect prototype

Table 2. Prototype Configuration and Hyperparameters

Parameter	Value
Input document format	DOCX document and DOCX SoT document
Embedding model	sentence-transformers/all-MiniLM-L6-v2

Embedding dimensionality	384
Prose chunk limit	max 250 words
k-NN retrieval depth	5
Per-pass similarity threshold	≥ 0.70
Candidate-pair combined-score threshold	≥ 0.65
Vector database	OpenSearch
Main vector index	doc_chunks
Persona profile index	persona_profiles
Review checkpoint index	review_checkpoints
Persona sampling size	5 representative chunks
Persona extraction model	gemini-3.7-flash
Change-generation model	gemini-3.1-pro-preview
Maximum batch-generation output tokens	65000

6. RESULTS AND DISCUSSION

The AutoDetect prototype was evaluated using document-processing statistics, retrieval results, reviewer decisions, manually labeled ground-truth data, and runtime measurements. The current document contained 78 chunks and the source-of-truth document contained 15 chunks, producing 93 embedded chunks in total. The system used the sentence-transformers/all-MiniLM-L6-v2 embedding model with 384-dimensional normalized vectors. Retrieval used $\text{TOP_K} = 5$, a per-pass similarity threshold of 0.70, and a combined candidate threshold of 0.65.

A. Retrieval Results

The bidirectional retrieval stage produced 23 high-confidence Edit or Context Change candidate pairs. It also produced 26 unknown pairs that were sent to the language model for further classification as Addition, Deletion, or Ignore. The highest combined similarity score among candidate pairs was 0.8346. Bidirectional retrieval required 11.8096 seconds.

The presence of 26 unknown pairs shows that semantic similarity alone cannot reliably determine whether content should be added, deleted, or ignored. These cases require language-model assessment and human review.

B. Review Results

The language-model stage produced 49 review suggestions, of which 28 had recognized Edit, Addition, or Deletion change types. Reviewers approved 25 of these suggestions and rejected 3, resulting in a manual approval rate of 0.893. No re-edit requests were recorded. The average similarity score of approved suggestions was 0.7668, while rejected suggestions had an average score of 0.7543. This small difference indicates that similarity score alone was not a strong predictor of reviewer approval.

Table 3 shows outcomes for the 28 items where the reviewer manually assigned a definite change type (Edit, Addition, or Deletion). The remaining 21 items were labeled Unsupported/Unknown by the reviewer and are excluded here.

Table 3. Reviewer-Assigned Change Type vs. Approval Outcome

Change Type	Review	Approve	Reject	Approval Rate
Edit	26	24	2	92.3%
Addition	0	0	0	-
Deletion	2	1	1	50.0%
Total	28	25	3	89.3%

Figure 2 shows reviewer outcomes for 28 recognized suggestions. Edit suggestions had a 92.3% approval rate, while Deletion suggestions had a 50.0% approval rate based on two items. No Addition suggestions were evaluated. Overall, 25 suggestions were approved and 3 were rejected, giving an 89.3% approval rate.

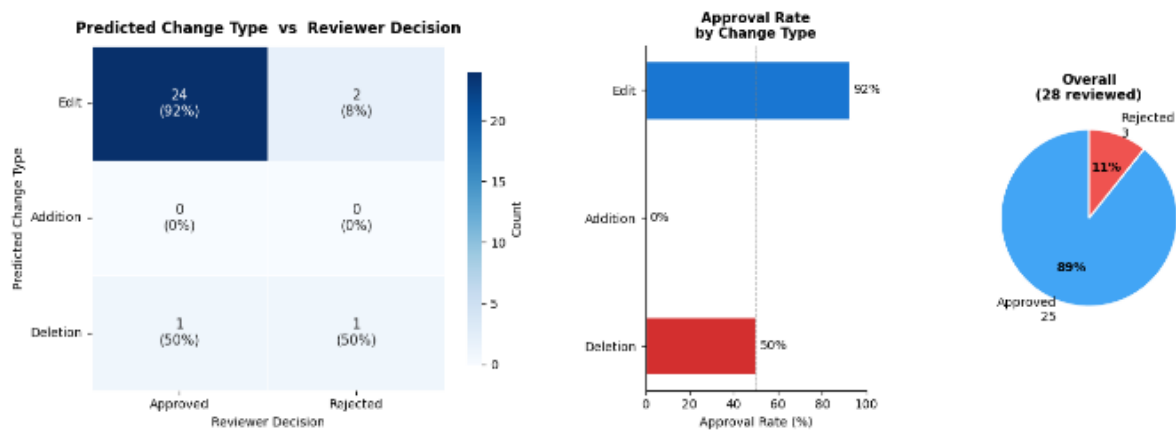


Fig. 2. Reviewer outcome matrix, approval rate by predicted change type, and overall reviewer decision distribution.

C. Ground-Truth Classification Results

A manually labeled ground-truth evaluation was conducted on 53 items. The evaluation used two classes: Edit and Ignore. It included prose, tables, bullets, and callouts, while headings and subheadings were excluded from evaluation.

Table 4 presents the classification metrics calculated from the manually labeled ground-truth data.

Table 4. Ground-Truth Classification Metrics

Metric	Result
Precision	0.9062
Recall	0.9014
Accuracy	0.9057
F1-score	0.9035

The per-class recall was 93.33% for Edit and 86.96% for Ignore. These results indicate that the system identified Edit cases effectively and also performed well when identifying content that did not require revision. The reviewer-based accuracy was 0.8929 for the 28 recognized reviewed items. This value is

based on reviewer approval and rejection decisions and should be interpreted separately from the manually labeled ground-truth accuracy of 0.9057.

D. Runtime Results

The execution-time measurements show the processing cost of each major pipeline stage. Table 5 presents the recorded time for parsing, embedding, indexing, retrieval, and LLM generation.

Table 5. Pipeline Runtime Measurements

Stage	Time
DOCX parsing	0.2111 s
Embedding generation	7.5597 s
OpenSearch indexing	0.5983 s
Bidirectional retrieval	11.8096 s
LLM generation	125.8133 s

Parsing and indexing required relatively little time. Embedding generation and bidirectional retrieval required more time because all chunks had to be converted into embeddings and compared against chunks from the opposite document. LLM generation required 125.8133 seconds for the recorded batch-generation run. Because the notebook generates pending review items in a batch request, this value represents total batch-generation time rather than the time required for one individual suggestion.

7. LIMITATIONS

The current prototype has several limitations. First, the system is limited to DOCX-based educational documents and does not yet support other common content formats such as PDF, web pages, slide decks, or plain-text repositories. The system also depends on the quality and completeness of the source-of-truth document. If the source of truth is incomplete, ambiguous, or poorly structured, the retrieval and revision suggestions may also be affected. Headings and subheadings are excluded from change detection, and DOCX export has placement and formatting limits.

8. FUTURE IMPROVEMENTS

Future enhancements can extend the system in several directions. Multi-document-type support can be added so that the system can process PDF files, web content, plain text, presentations, and other educational formats in addition to DOCX. Multi-content-type support can also be improved by handling images, diagrams, mathematical expressions, formulas, code blocks, charts, and complex tables more accurately. Future versions should use a larger benchmark dataset, expanded manually labeled ground-truth evaluation, more complete runtime measurement, and broader reviewer-based evaluation. The system can also be improved through better deletion validation, adaptive similarity thresholds, more robust persona modeling, and integration with learning management systems or document management platforms.

9. CONCLUSIONS

This paper presented the AutoDetect Content Change System, a prototype for detecting and revising outdated educational content using structure-aware DOCX parsing, semantic embeddings, OpenSearch-

based bidirectional retrieval, LLM-assisted revision generation, persona-aware drafting, and human review. The system addresses the limitations of manual comparison and keyword-based search by identifying meaning-level relationships between a current document and a source-of-truth document. The experimental results demonstrate that the prototype can retain document structure during parsing, generate semantic representations, retrieve candidate changes, and prepare revision suggestions for reviewer evaluation. Although the current implementation is limited to DOCX documents and requires evaluation on larger manually labeled datasets, the proposed approach provides a practical foundation for controlled educational content maintenance. By combining automation with human oversight, the system supports more reliable, style-consistent, and reviewer-approved updates to educational materials.

10. REFERENCES

1. Alvi, Faisal, Hasan Algafr, and Naif Alqahtani. "Style Change Detection using Discourse Markers." Clef (working notes). 2022.
2. Jiang, Xinyin, et al. "Style Change Detection: Method Based On Pre-trained Model And Similarity Recognition." Clef (working notes). 2022.
3. Yang, Yinfei, et al. "Multilingual universal sentence encoder for semantic retrieval." Proceedings of the 58th annual meeting of the Association for Computational Linguistics: system demonstrations. 2020.
4. Boddupally, Hema Latha. "Intelligent semantic retrieval pipelines driving scalable, context-aware, and high-fidelity knowledge management capabilities across complex enterprise application landscapes." Context-Aware, and High-Fidelity Knowledge Management Capabilities Across Complex Enterprise Application Landscapes (August 30, 2023) (2023).
5. Wan, Mengting, et al. "Tnt-llm: Text mining at scale with large language models." Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining. 2024.
6. Kumar, Sushant, et al. "Applications, challenges, and future directions of human-in-the-loop learning." IEEe Access 12 (2024): 75735-75760.