iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IV **Month of publication:** April 2026

DOI: https://doi.org/10.22214/ijraset.2026.79681

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

ML-Based Automated Handwritten Answer Sheet Evaluation: System Design, Implementation, and Real-World Deployment

Akshat Andhale[1], Rohit Chaudhari[2], Pratik Derle[3], Om Jadhav[4], Kunal Ahire[5]
Department of Information Technology, MET's Institute of Engineering, Savitribai Phule Pune University

Abstract: ***This paper presents a detailed implementation study of a fully deployed, cloud-native, ML-based system for automated answer sheet evaluation, designed specifically for Indian higher-education institutions. Building on a prior theoretical proposal, this work documents the architectural decisions, algorithmic choices, and engineering trade-offs that arose during real-world deployment across a multi-tenant institutional hierarchy encompassing colleges, branches, classes, subjects, and students. The system accepts scanned handwritten student PDFs, rasterises each page at 3× scale, produces three preprocessing variants per page (original, strong-contrast binarised, and notebook-clean binarised), selects the highest-scoring variant, and submits it to the Google Cloud Vision API. The resulting raw OCR text passes through a four-layer correction pipeline: Unicode glyph normalisation, domain-specific OCR word-error correction, structural regex repair, and a final refinement step via Google Gemini 2.0 Flash. The corrected text is then parsed into a structured question–part hierarchy by a deterministic finite-state parser designed to tolerate OCR-induced label corruption. Scoring relies on a hybrid two-component engine that combines a multi-metric lexical scorer — incorporating the Jaccard index, bidirectional information-weighted fuzzy token coverage, character tri-gram similarity, and a containment score — with dense semantic similarity derived from Google Gemini Embedding-001 (768-dimensional vectors, cosine similarity, re-normalised to [0,1]), weighted 20% lexical and 80% semantic by default. A configurable minimum-credit heuristic ensures that semantically valid but lexically divergent answers are not inadvertently assigned zero marks. The backend is a Node.js 20/Express 4 REST API with all persistent data stored in Supabase (PostgreSQL with managed object storage), accompanied by a vanilla-JS web application offering role-separated interfaces for administrators, teachers, district-level uploaders, and students. On a validation set of 16 student scripts across three subjects, the hybrid engine achieves a Pearson r = 0.91 against teacher-assigned marks and a within-±1-mark accuracy of 81.2% — a 20-percentage-point improvement over lexical-only grading. Average end-to-end processing time is approximately 22 seconds per six-page answer sheet, enabling a 50-student cohort to be evaluated in under 20 minutes.***

Index Terms: ***Automated Short-Answer Grading, Handwritten Text Recognition, Google Cloud Vision OCR, Gemini Embeddings, Semantic Similarity, Fuzzy Token Matching, Multi-Tenant Web Application, Supabase, Node.js, Role-Based Access Control, Educational Technology***

I. INTRODUCTION

Manual grading of descriptive handwritten examination scripts has remained the dominant evaluation method in Indian higher education for more than a century. Despite well-documented shortcomings in consistency, turnaround time, and scalability, institutional inertia and the lack of practical automated alternatives have preserved the status quo. This challenge has become more pressing as enrolment in technical and professional programmes continues to grow: a single faculty member may be responsible for grading several hundred scripts within a window of three to five days, a constraint that undermines the quality of feedback and delays important academic decisions.

Three converging technological advances have made the automated grading of handwritten scripts increasingly viable. First, cloud-based Optical Character Recognition (OCR) services — particularly Google Cloud Vision — have reached character error rates below 10% on moderately legible handwriting, compared to 25–40% for locally-deployed Tesseract on equivalent inputs. Second, transformer-based language model embeddings (BERT, Sentence-BERT, and more recently Google Gemini Embedding-001) generate dense vector representations of short text passages that capture semantic equivalence far more reliably than term-frequency approaches. Third, cloud-managed databases and storage services such as Supabase allow small development teams to deploy production-grade, multi-tenant applications without having to manage their own database infrastructure.

©IJRASET: All Rights are Reserved

The authors' earlier work proposed a complete conceptual architecture for ML-based automated evaluation, drawing on the literature to justify choices in OCR pipeline design, semantic similarity metrics, and system modelling. That paper, however, remained at the design level and did not produce a working system. The present paper reports the result of the subsequent engineering effort: a running, institutionally deployed web application that processes real scanned handwritten answer sheets, assigns marks for each question part, generates digitised PDF transcripts, and presents results through role-appropriate web interfaces.

The primary contributions of this work are as follows:

1) A production-quality Node.js REST API (≈ 3,600 lines) integrating Google Cloud Vision, Google Gemini 2.0 Flash, Gemini Embedding-001, pdfjs-dist, and Supabase, deployed without external tooling or build steps.

2) A novel three-variant OCR preprocessing strategy (original, strong-contrast, notebook-clean binarisation) with an information-theoretic best-candidate selector.

3) A five-component lexical similarity metric (Jaccard, bidirectional information-weighted fuzzy token coverage, character n-gram similarity, containment score) rigorously combined with Gemini dense-vector semantic similarity.

4) A minimum-credit heuristic that applies asymmetric thresholds based on answer length to prevent under-scoring of semantically correct but lexically divergent short responses.

5) A multi-tenant institutional hierarchy (college → branch → class → subject → student) with four distinct user roles, fine-grained access control, and district-scoped uploader permissions.

6) A detailed account of engineering challenges encountered during deployment and the concrete solutions adopted, intended as a reproducible reference for practitioners in educational technology.

The remainder of this paper is organised as follows. Section 2 reviews related work. Section 3 describes the system architecture. Section 4 details the OCR and image processing pipeline. Section 5 covers NLP preprocessing and question parsing. Section 6 analyses the hybrid scoring engine. Section 7 describes the multi-tenant access control design. Section 8 discusses implementation challenges. Section 9 presents experimental results. Section 10 compares the system with prior work. Section 11 addresses limitations and future directions. Section 12 concludes.

II. RELATED WORK

A. *Automated Short Answer Grading*

Automated short-answer grading (ASAG) has been an active research area since the early 2000s. Early systems such as c-rater and BETSY relied on hand-crafted syntactic features and pattern-matching rules, which performed reasonably well on curated datasets but required considerable per-domain engineering effort. The shift to vector-space models (VSM) with TF-IDF representations simplified deployment, but these approaches failed to capture semantic paraphrase — a critical weakness when students express correct ideas using non-standard vocabulary.

Word embedding approaches (Word2Vec, GloVe) partially addressed the paraphrase problem by representing words as dense vectors in a learned semantic space, enabling softer lexical matching. However, these models produce context-independent embeddings and cannot disambiguate polysemous words in different grammatical roles. Transformer-based models — BERT, RoBERTa, and their sentence-level derivatives (Sentence-BERT) — overcame this limitation through bidirectional contextual encoding and set new benchmarks on ASAG tasks including SemEval, ASAP, and the Mohler dataset. Haller et al.'s 2022 survey provides a comprehensive review of deep-learning ASAG methods, noting that Sentence-BERT with cosine similarity consistently outperforms earlier approaches across multiple datasets.

B. *Handwritten Text Recognition for Grading*

Most ASAG research assumes typed or digital input; far fewer studies address the full pipeline starting from scanned handwritten scripts. The IAM Handwriting Database and its derivatives have enabled the training of CRNN-based handwriting recognisers that achieve character error rates below 5% on constrained vocabularies. Exam answer sheets, however, present additional challenges not found in standard benchmark datasets: ruled-line background noise, variable ink density, a mix of printed and cursive writing, and structural artefacts specific to multi-question formatted sheets. Bansal et al. evaluated handwritten answers using a Tesseract + BERT pipeline, reporting 72% within-tolerance accuracy on a small dataset. Pawar proposed a custom CNN-LSTM OCR integrated with TF-IDF scoring, achieving 68% accuracy but requiring a domain-specific training corpus. To the best of our knowledge, cloud-based OCR APIs have not previously been systematically evaluated in the grading context, despite offering substantially lower character error rates than offline engines on realistic handwriting.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

C. Gemini Embeddings for Semantic Similarity

Google's Gemini Embedding-001 model, released in 2024, produces 768-dimensional embeddings optimised for semantic similarity tasks via a task Type: SEMANTIC_SIMILARITY instruction. This distinguishes it from earlier Google Universal Sentence Encoder models, as Gemini Embedding-001 supports instruction-following for task-specific pooling. To our knowledge, no prior published work has evaluated Gemini Embedding-001 as a scoring signal in the ASAG context, making the present study the first to characterise this model's behaviour in a grading setting.

III. SYSTEM ARCHITECTURE

A. Architectural Overview

The system follows a conventional three-tier web application architecture, with substantial AI augmentation in the backend processing layer. All backend logic is contained in a single server's file (~3,600 lines) written using Node.js ES modules. This was a deliberate design choice: concentrating all logic in one deployable file eliminates build-tooling dependencies and simplifies deployment to low-footprint servers. The entire system can be started with a single node server's command once the required environment variables are set.

TABLE I
TECHNOLOGY STACK — COMPLETE DEPENDENCY INVENTORY

Tier	Technology	Version	Role
Runtime	Node.js	20 (ESM)	Server-side JavaScript execution
Web framework	Express.js	4.18.2	REST routing, middleware, static serving
File upload	Multer	1.4.5-lts.1	Multipart PDF ingestion
OCR	Google Cloud Vision API	v1 REST	DOCUMENT_TEXT_DETECTION per page
LLM correction	Google Gemini 2.0 Flash	v1beta REST	OCR error correction via generation
Embeddings	Google Gemini Embedding-001	v1beta REST	768-dim semantic similarity vectors
PDF rendering	pdfjs-dist (legacy)	5.5.207	Server-side PDF-to-canvas rasterisation
Canvas API	@napi-rs/canvas	0.1.95	Native canvas binding for Node.js
PDF generation	PDFKit	0.17.2	Digitised answer transcript generation
Database	Supabase / PostgreSQL	Managed cloud	Relational data + JSON evaluation store
Object storage	Supabase Storage	Managed cloud	PDF file persistence, signed URL access
Frontend	Vanilla HTML/CSS/JS	No framework	Role-separated single-page interfaces
Auth	Custom token + HttpOnly cookie	In-process Map	Session management for four user roles

B. Request Lifecycle

Every incoming HTTP request passes through a two-stage middleware chain. The first stage — requireRoleApi() for API endpoints or requireRolePage() for HTML routes — extracts the session token from the auth_token HttpOnly cookie or the Authorization header, looks it up in the in-process authSessions Map, and verifies that the authenticated user's role matches the permitted set. The second stage, requireDatabase(), responds with HTTP 503 if the Supabase client was not successfully initialised (i.e., required environment variables are missing), providing a meaningful error message rather than allowing the server to crash.

Approximately 40 REST endpoints are organised into seven functional groups: authentication, organisational hierarchy CRUD, user management, answer key management, answer sheet upload management, evaluation execution, and results retrieval. All endpoints return JSON; file-serving endpoints return binary content with appropriate MIME types.

C. *Database Schema*

The PostgreSQL schema (defined in supabase_schema.sql) reflects a five-level containment hierarchy that mirrors the actual institutional structure of Indian higher education: colleges → branches → classes → subjects → students, with teachers and uploaders as cross-cutting entities. All primary keys are UUIDs generated by gen_random_uuid() (which requires the pgcrypto extension). Eleven composite indexes are defined to support the filtering patterns the application relies on.

TABLE II.
POSTGRESQL SCHEMA — ALL TABLES WITH THEIR KEY COLUMNS, CONSTRAINTS, AND PURPOSES

Table	Key Columns	Constraints/Indexes	Purpose
colleges	id, name, address, district, state, pincode	name UNIQUE; idx on district	Root institutional entity with geographic metadata
branches	id, college_id, name	UNIQUE(college_id, name); idx on college_id	Academic departments within a college
classes	id, branch_id, name	UNIQUE(branch_id, name); idx on branch_id	Year/section within a branch
subjects	id, class_id, name, code	idx on class_id	Academic subject linked to a class
students	id, class_id, name, roll_number	roll_number UNIQUE; idx on class_id	Individual student with class assignment
teachers	id, username, password, name	username UNIQUE	Teacher credentials with hashed password
teacher_classes	teacher_id, class_id	UNIQUE(teacher_id, class_id); composite idx	Many-to-many: teacher ↔ class assignments
teacher_subjects	teacher_id, subject_id	UNIQUE(teacher_id, subject_id); composite idx	Many-to-many: teacher ↔ subject assignments
uploaders	id, username, password, district	username UNIQUE; idx on district	District-scoped data-entry operators
answer_keys	id, subject_id, structured (JSONB), pdf_path	subject_id UNIQUE (1:1)	Structured model answer store for each subject
evaluations	id, subject_id, student_id, class_id, comparison (JSONB)	idx on subject+student; idx on class_id	Full evaluation result as structured JSONB payload

IV. OCR & IMAGE PROCESSING PIPELINE

A. *PDF Rasterization*

Student answer sheets are submitted as PDF files via the POST /answer-sheet-uploads endpoint. The evaluation pipeline begins by rasterising each PDF page using pdfjs-dist in legacy (Node.js-compatible) mode. Each page is rendered onto an @napi-rs/canvas instance at a scale factor of 3.0, producing an effective resolution roughly three times the native PDF page resolution. For a standard A4 page at 72 DPI (the default PDF coordinate system), this yields a 2480 × 3508 pixel canvas — comfortably above the 300 DPI minimum recommended for handwriting OCR. The canvas is initialised with a white background fill prior to rendering, ensuring consistent behaviour for PDFs with transparent backgrounds. To accommodate up to 15 pages per uploaded document — a practical limit that covers all realistic Indian university examination formats — the rasterisation loop processes pages sequentially. Parallelisation was deliberately deferred due to the memory pressure caused by handling multiple large canvases concurrently in Node.js, a known limitation of the @napi-rs/canvas native binding in server environments; this is discussed further in Section 8.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

B. Three-Variant OCR Processing Strategy

One of the key contributions of this implementation is a three-variant preprocessing strategy applied to each rasterised page before OCR submission. Rather than submitting a single image, the system generates three distinct contrast variants and selects the one that produces the highest-scoring OCR result:

1) Original: The raw pdfjs-rendered canvas, submitted without modification. These variant preserves colour and gradient information and performs best for printed or high-contrast typed content.

2) Strong contrast binarisation: Applies a luminance-weighted grayscale conversion (L = 0.299R + 0.587G + 0.114B), followed by contrast boosting with a factor of 1.8 centred at 128, then hard thresholding at 185 to produce black-and-white output. This enhances dark ink strokes against light backgrounds and works best for freshly written ballpoint-pen text on white paper.

3) Notebook-clean binarisation: Applies the same grayscale conversion but uses a two-threshold step function: pixels with luminance > 210 map to white, pixels < 155 map to black, and all others also map to white. This approach aggressively removes the light-blue ruled lines common in Indian examination notebooks while preserving dark ink strokes — a pattern absent from Western handwriting recognition benchmarks but ubiquitous in Indian institutional settings.

All three variants are submitted to the Google Cloud Vision DOCUMENT_TEXT_DETECTION endpoint. The pickBestOcrText() selector ranks candidates by a composite information score:

score(text) = len(text) + 30 × count(Q-headers) + 10 × count(part-labels) + 2 × count(newlines)

This scoring function rewards OCR output that contains structured question-number tokens (Q1, Q2) and sub-part labels (a), b)), which are the primary structural anchors used by the downstream parser. Text length acts as a tiebreaker between candidates with equal structural richness.

C. Google Cloud Vision API Integration

Each image variant is submitted to the Vision API via the POST https://vision.googleapis.com/v1/images:annotate REST endpoint, using featureType: DOCUMENT_TEXT_DETECTION and languageHints: ["en"]. The DOCUMENT_TEXT_DETECTION feature is preferred over plain TEXT_DETECTION because it returns a fullTextAnnotation object that preserves paragraph, block, and word-level bounding box structure — information that helps the downstream parser reconstruct answer boundaries across OCR-disjointed words. The system first reads fullTextAnnotation.text; if this is absent (as can happen with lightly printed or low-confidence pages), it falls back to textAnnotations[0]. description, the flat concatenation of all recognised strings.

D. Searchable PDF Fast Path

Before invoking Vision OCR, the system applies a fast-path check using extractTextFromSearchablePdf(), which attempts direct text extraction from the PDF binary via pdfjs-dist's getTextContent() API. When a student submits a PDF generated by a document scanner that also embeds searchable text — a common feature of modern mobile scanning apps — this path avoids an expensive cloud API call entirely. The extracted text is then validated by looksLikeStructuredAnswerText(), which checks for the presence of at least one question-number anchor (Q1, 1., 1)) before accepting the result. If this check fails, the pipeline falls through to Vision OCR. A secondary custom fallback, extractTextFromPdfBuffer(), parses raw PDF Tj and TJ text operators directly from the binary stream using zlib inflation, providing a last-resort extraction path for PDFs whose getTextContent() output is empty but whose binary contains embedded glyph codes. This three-tier extraction strategy (pdfjs text → binary operators → Vision OCR) maximises use of locally available text before incurring API costs.

V. NLP PREPROCESSING & QUESTION PARSING

A. Four-Layer OCR Correction Stack

Raw Vision OCR output for handwritten answer sheets routinely contains several categories of error that must be corrected before the text can be reliably parsed or scored:

1) Glyph substitution: Vision's character recogniser misreads specific handwritten letterforms. The most frequent substitutions observed in the test dataset include: rn → m, 0 → o, 1 → l/i, 5 → s, 8 → b.

2) Domain word errors: Discipline-specific vocabulary is frequently mangled: "operatim" → "operating", "sycten" → "system", "softuare" → "software". A 25-entry lookup table covers the most common Information Technology domain errors observed empirically.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

3) Structural label corruption: Question numbers and part labels are the most critical structural tokens. OCR frequently reads Q1 as O1, Q2 as 92 (the '9' prefix is a common OCR artefact for 'Q'), and part labels a) as 9), 4).

4) Notebook header noise: Indian examination notebooks print repeated header text ("Page No.", "Date", "CLASSMATE") on every page. These tokens are filtered by the isNoiseLine() predicate.

The OCR correction pipeline is implemented as a four-layer processing stack designed to progressively refine the raw extracted text.

a) Layer 1: normalizeOcrGlyphs(text)
Replaces rn→m, vv→w, li→h, ln→in, fi→h, fl→h, compute r→computer, cuTrently→currently, etc. (25 rules).

b) Layer 2: repairMergedPartHeader(line)
Splits OCR-merged tokens such as ‘aManages’ into ‘a) Manages’ using the pattern: single-char + UpperCase boundary.

c) Layer 3: normalizeOcrLine(line)
Corrects question-number prefixes: O[0-9] → Q[0-9], 0[0-9] → Q[0-9], 9[0-9] → Q[0-9], Ql → Q1, Q i → Q1.

d) Layer 4: correctStructuredOcrWithGemini(text)
Invokes Gemini 2.0 Flash with temperature=0.1, topP=0.8, maxOutputTokens=4096. The prompt instructs the model to ‘correct OCR spelling mistakes, preserve structure exactly, do not paraphrase’. This layer is skipped if the API is in cooldown, in which case the Layer 3 output is used directly.

B. Deterministic Question-Part Parser

The normalizePartLabel() function handles a critical OCR edge case: digit–letter substitution in part labels. This function maintains an ocrPartMap (‘9’→‘a’, ‘4’→‘a’, ‘8’→‘b’, ‘6’→‘c’, ‘e’→‘c’) and also applies expectation-based correction: if the parser expects label ‘b’ (because ‘a’ is already populated) but OCR produces ‘a’ again, the duplicate is normalised to ‘b’. This sequential expectation logic tolerates label ambiguity that pure rule-based correction cannot resolve.

When line-based parsing collapses all content into a single question — a known failure mode for very densely handwritten scripts where Vision OCR omits newlines — the parser automatically falls back to extractQuestionsByAnchors(), an anchor-based segmentation approach that locates question boundaries using regex anchors on the linearised text rather than line breaks.

The corrected OCR text is parsed into a structured { Q1: { parts: { a: '...', b: '...' } }, Q2: { ... } } object by extractQuestions(), a deterministic finite-state parser operating over the line-tokenised input. The parser maintains two state variables (currentQ and currentPart) and transitions through the following logic per line:

FOR each line in lines:
    IF line matches question header regex:
        → Transition to new question state Q_n
        → If trailing text after Q-header matches part regex:
            → Also set currentPart
    ELSE IF line matches part header regex:
        → If no currentQ: infer next Q number
        → If currentQ already has this part label: infer next Q
        → Set currentPart
    ELSE:
        → Append to currentPart text (if active) or question intro

C. Student Answer Repair Against Teacher Key

After structured extraction, each student answer undergoes repairStudentAnswerAgainstTeacher(): a targeted fuzzy spell-correction step that replaces student tokens with the closest teacher-key token when their fuzzy similarity exceeds 0.78. This step operates only on alphanumeric tokens of length ≥ 3, and only when the length difference between the student and teacher tokens is ≤ 4 characters, preventing over-correction of legitimate alternative vocabulary. The repair improves downstream lexical matching by resolving residual OCR character substitutions that the domain word-map did not cover.

VI. HYBRID SCROING ENGINE

A. Architectural Overview

The scoring engine (compareStructuredAnswers()) operates on the paired structured representations of student and teacher answers. For each teacher-defined question Q_n with parts {a, b, c, …}, it computes a final similarity score for each (student-part, teacher-part) pair and converts that score to marks proportionally. The evaluation result is stored as a JSONB payload in the evaluations table, containing per-part similarity percentages at three granularities (lexical, semantic, and blended), awarded marks, and raw OCR text, enabling full auditability.

B. Lexical Similarity

The lexical similarity function similarityScore() combines five sub-metrics to produce a single scalar in [0, 1]. Each sub-metric captures a distinct aspect of token-level overlap:

TABLE III
LEXICAL SIMILARITY SUB-METRICS AND THEIR ROLES

Sub-Metric	Formula/Description	Weight/Role
Jaccard index	common_unique_tokens / union_unique_tokens	Global set-level overlap; penalises both missing and extra tokens.
Teacher coverage (strict)	common_unique_tokens / |teacher_token_set|	Rewards completeness: how much of the teacher's vocabulary does the student cover?
Min bidirectional coverage	min(student_cov, teacher_cov)	Conservative overlap: both directions must agree. Penalises irrelevant verbosity.
Fuzzy token coverage (weighted)	Σ(teacher_token_weight × best_fuzzy_match) / Σ(weights)	Tolerates OCR character substitutions; weight = min(1.6, 0.8 + len/8).
Character tri-gram similarity	2 × |A_trigrams ∩ B_trigrams| / (|A_trigrams| + |B_trigrams|)	Sub-word overlap; catches morphological variants (plurals, tense).
Containment score	max(|left| / |right|, |right| / |left|) if one is substring of other	Exact substring containment; threshold length > 18 chars.

C. Semantic Similarity via Gemini Embedding

When a Gemini API key is configured, semantic similarity is computed by getSemanticSimilarityBatch() using the gemini-embedding-001 model via the batch embedding endpoint:

1) API: POST https://generativelanguage.googleapis.com/v1beta/models/gemini-embedding-001:batchEmbedContents
2) Task type: SEMANTIC_SIMILARITY
3) Embedding dimension: 768

VII. MULTI-TENANT ROLE-BASED ACCESS CONTROL

A. User Roles and Permissions

The system implements a four-role RBAC model reflecting the actual stakeholder structure of Indian institutional examinations:

TABLE IV
ROLE BASED ACCESS CONTROL

Role	Storage	Scope	Key Capabilities
Administrator	Environment vars (single account)	Full system	All CRUD; bulk evaluation trigger; config management; college/branch/class/subject/teacher/student/uploader management
Teacher	teachers table (admin-created)	Assigned classes and subjects	View evaluation results for assigned subjects; view answer key status; delete own evaluation records

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

Uploader	uploaders table (admin-created)	District-scoped colleges → classes	Submit student answer sheets; view upload status; cascade-filtered by district
Student	students table (roll number)	Own records only	View own evaluation results and digitised answer PDF; read-only portal

B. Frontend Interface Architecture

TABLE V
FRONTEND PAGES

Page	Role	Key Functionality
login.html	All	Role-selector dropdown, credential form, cookie-based session initiation
admin.html	admin	Subject selector; bulk evaluation trigger; evaluation config display (weights, min-credit flag); result summary table
admin-hierarchy.html	admin	College → branch → class CRUD with cascading selects; address/district/state/pincode management
admin-users.html	admin	Teacher and uploader account CRUD; class/subject assignment for teachers; district assignment for uploaders
admin-students.html	admin	Student enrolment; class assignment; roll number management; bulk student list
admin-keys.html	admin	Answer key PDF upload; OCR extraction status; per-question structured preview; question count display
teacher.html	teacher	Class/subject selector (scoped to assignments); results table with per-question breakdown; answer key status panel
uploader.html	uploader	Cascading selectors (college → branch → class → subject → student); PDF file picker; upload status feedback; district badge
student.html	student	Roll-number login; personal result cards; per-question marks display; downloadable digitised answer PDF
digitizer.html	admin	Manual OCR testing tool; single PDF upload; raw OCR text viewer; structured parse output debugger

VIII. IMPLEMENTATION CHALLENGES AND ENGINEERING SOLUTIONS

A. OCR Structural Parsing Robustness

The most significant challenge during implementation was achieving reliable structured parsing of raw Vision OCR output. Vision OCR produces a linearised text stream without semantic markup; reconstructing the Q1/Q2/a)/b)/c) hierarchy from this stream requires tolerating several types of structural corruption simultaneously:

1) Missing newlines: Densely handwritten scripts sometimes produce a single continuous text block rather than line-separated content. The anchor-based fallback parser handles this case by locating question boundaries in the linearised text.

2) Digit–letter confusion in labels: The most disruptive error pattern is OCR reading 'Q2' as '92'. The normalizeQuestionNumber() function detects two-digit numbers starting with '9' and converts them: 92 → 2, 93 → 3. This covers the vast majority of Q-label corruption observed empirically

3) Duplicate part labels: When a new question begins immediately after a sub-part without a blank line, OCR sometimes reassigns the first sub-part of the new question the same label as the last sub-part of the previous question. The shouldInferNextQuestionFromDuplicatePart() function detects this and creates a new question entry automatically

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

4) Page-boundary answer continuation: When an answer spans two pages, the page separator inserted by the rasterisation loop ('\n') must not be treated as a question boundary. The parser's state machine correctly handles this because it only transitions to a new question on an explicit Q-header match, not on a blank line alone.

B. Multi-Page PDF Memory Pressure

Rasterising a 15-page PDF at 3× scale with three variants per page requires allocating 45 large canvas objects in sequence. In Node.js, the @napi-rs/canvas native binding does not expose an explicit canvas disposal API; memory is reclaimed only by the garbage collector. Under sustained load with several simultaneous evaluations, heap size was observed to spike above 2 GB in testing. The adopted solution was to process pages sequentially (rather than in parallel) within each document, ensuring that at most three canvas objects are live at any one time. A maxPages = 15 cap provides a hard upper bound on memory consumption per evaluation. Parallelism was traded for memory safety; explicit canvas buffer reuse or worker-thread isolation would be required to enable safe parallelisation in the future.

C. Supabase Storage Signed URL Lifecycle

All uploaded PDFs (teacher answer keys and student answer sheets) are persisted in Supabase Storage. The application does not store raw file paths as public URLs — instead, it generates signed URLs on demand with a configurable TTL (default 86,400 seconds = 24 hours). Every call to GET /answer-sheet-uploads and GET /results regenerates signed URLs for all returned records. This adds a small latency overhead (one Supabase API call per file reference per request) but eliminates stale-URL errors in the frontend and avoids caching signed URLs in the database, where they would become invalid after expiry without a background refresh job.

D. Evaluation Idempotency and Re-evaluation

The POST /evaluate-uploads-bulk endpoint processes all uploads for a given class/subject pair. If an evaluation record already exists for a (student, subject) pair, re-evaluation overwrites it by calling upsertEvaluationDb() rather than insertEvaluationDb(). The upload status is updated from 'uploaded' to 'evaluated' in the answer_sheet_uploads table after each successful evaluation. This enables administrators to re-trigger bulk evaluation after updating the teacher answer key or changing scoring configuration without accumulating duplicate evaluation records.

E. Digitised Answer Transcript Generation

As a transparency measure, the system generates a digitised answer transcript PDF for each evaluated student using PDFKit. The transcript presents the structured OCR output — question numbers, part labels, and extracted answer text — in a clean formatted layout with the student name, roll number, and subject as a header. This transcript is stored in Supabase Storage and linked in the evaluation result, allowing teachers to verify OCR accuracy and students to review how their handwriting was interpreted by the system. The generation step adds approximately 0.3 seconds per student and is performed inline within the bulk evaluation loop.

IX. EXPERIMENTAL EVALUATION

A. Dataset

A validation dataset was manually assembled comprising 16 handwritten student answer scripts across three first-year engineering subjects: Computer Organisation and Architecture (6 scripts), Engineering Mathematics (5 scripts), and Fundamentals of Information Technology (5 scripts).

Each script contained 5 questions with 2–3 sub-parts each, producing 240 individual (student-part, teacher-part) scoring instances. Scripts were scanned at 300 DPI using a commercial flatbed scanner and submitted as PDF files. Ground-truth marks were assigned by the subject teachers independently of the automated system before evaluation.

The dataset is intentionally small, reflecting the practical constraint of obtaining fully annotated ground-truth evaluations from busy institutional teachers. Statistical confidence intervals should therefore be interpreted conservatively; the reported metrics are intended as directional indicators rather than definitive performance bounds. A larger validation study involving 200+ scripts across diverse subjects is identified as priority future work.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

B. OCR Accuracy

The performance of the OCR pipeline was evaluated across multiple processing stages using Character Error Rate (CER) as the primary metric. The raw output obtained from the Google Vision DOCUMENT_TEXT_DETECTION API achieved a mean CER of 8.3% with a standard deviation of ±3.1%, serving as the baseline.

After applying Layers 1–3 of rule-based correction, the mean CER improved to 6.1% (±2.4%), representing a relative improvement of 26.5% over the baseline. Further refinement using Layer 4, which incorporates Gemini-based correction, reduced the CER to 4.1% (±1.9%), corresponding to a total improvement of 50.6%.

When considering only structured label tokens such as question headers and part labels, the CER decreased further to 3.2% (±1.6%), achieving the highest relative improvement of 61.4%. In contrast, free-form answer text exhibited a CER of 5.7% (±2.3%), corresponding to a relative improvement of 31.3%, reflecting the greater difficulty of correcting unconstrained textual content.

C. Scoring Accuracy

The scoring performance of different configurations is evaluated using within ±1 mark accuracy, mean absolute error, and correlation metrics. The lexical-only configuration, based on Jaccard similarity and token coverage, achieves a within-tolerance accuracy of 61.3% with a mean absolute error of 1.42 marks, and correlation values of Pearson $r = 0.74$ and Spearman $\rho = 0.71$.

Introducing the minimum credit rule to the lexical approach improves the accuracy to 66.8% and reduces the mean absolute error to 1.19 marks, with corresponding correlation values of $r = 0.77$ and $\rho = 0.75$. The semantic-only configuration using Gemini Embedding-001 further improves performance, achieving 72.8% accuracy, a mean absolute error of 0.98 marks, and correlation values of $r = 0.83$ and $\rho = 0.82$.

The hybrid configuration, combining lexical and semantic components with weights $w_L = 0.2$ and $w_S = 0.8$, achieves a within-tolerance accuracy of 76.1% and a mean absolute error of 0.89 marks, with correlation values of $r = 0.87$ and $\rho = 0.86$. The full system, incorporating both hybrid scoring and the minimum credit rule, achieves the best performance with 81.2% accuracy, a mean absolute error of 0.76 marks, and strong correlation values of Pearson $r = 0.91$ and Spearman $\rho = 0.89$.

D. Processing Throughput

The processing throughput of the system was evaluated by measuring the average time required for each stage of the evaluation pipeline. PDF rasterisation, which generates three variants for up to 15 pages, requires approximately 5.8 seconds per sheet and operates sequentially, scaling with page count. The Google Vision OCR stage, which processes three variants per page, takes approximately 8.4 seconds and represents the dominant latency source due to its network-bound nature.

The rule-based OCR correction stages (Layers 1–3) and the question parsing and answer repair steps both require less than 0.1 seconds and are computationally negligible, being CPU-bound and deterministic. The Gemini-based correction (Layer 4) takes approximately 3.4 seconds per sheet and involves a single API call, which may be skipped during cooldown periods. Additionally, Gemini batch embedding requires approximately 4.8 seconds per subject, but this cost is shared across all students and contributes only a marginal overhead per individual evaluation. Subsequent stages such as scoring and JSONB serialisation take approximately 0.3 seconds, while digitised PDF generation using PDFKit also requires around 0.3 seconds per student. Database write and storage upload operations via Supabase take approximately 1.6 seconds, involving two round trips for evaluation insertion and PDF upload.

Overall, the total end-to-end processing time for a typical six-page handwritten answer sheet is approximately 24 seconds, including all input/output operations. For a cohort of 50 students evaluated for a single subject, the system completes bulk evaluation in approximately 18–22 minutes. The Vision OCR stage accounts for the largest proportion of processing time, contributing around 35% of the total latency. A key optimisation opportunity lies in parallelising Vision OCR requests across pages. While the current sequential processing approach ensures memory stability, parallel submission could reduce OCR latency from 8.4 seconds to approximately 2–3 seconds for a six-page document, thereby lowering the total processing time to approximately 12–15 seconds per sheet.

X. COMPARISON WITH RELATED SYSTEMS

The proposed system is compared with existing automated grading approaches in terms of OCR methodology, scoring technique, deployment status, multi-tenancy support, and evaluation accuracy. The present system employs a combination of Google Cloud Vision, Gemini Flash, and a three-variant preprocessing strategy, along with a hybrid scoring mechanism that integrates Gemini Embedding-001 (80%) and multi-metric fuzzy lexical scoring (20%). It is fully deployed as a Node.js and Supabase-based web application with a four-role role-based access control (RBAC) system and supports a five-level multi-tenant institutional hierarchy. The system achieves a within ±1 mark accuracy of 81.2%.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

In comparison, the system proposed by Bansal et al. (2025) uses Tesseract OCR combined with BERT cosine similarity but remains a research prototype without deployment or multi-tenant support, achieving an accuracy of 72%. Similarly, Pawar (2025) employs a custom CNN-LSTM OCR with TF-IDF and cosine similarity for scoring, but lacks deployment and multi-tenant capabilities, achieving an accuracy of 68%. Bonthu et al. (2021) consider only typed input and use a BiLSTM classifier, achieving an accuracy of 74% without deployment or multi-tenant support. The prior work by the same team proposes a CNN-BiLSTM OCR combined with BERT and SBERT scoring but remains purely conceptual and lacks implementation and evaluation metrics.

The principal differentiators of the present system relative to prior work include:

1) End-to-end deployment on real scanned handwritten PDFs within a production-grade web application.
2) The use of a three-variant OCR preprocessing strategy that improves structured label recognition, particularly for Indian notebook formats.
3) The adoption of Gemini Embedding-001 as the semantic backbone, which has not been evaluated in prior grading systems.
4) The implementation of a multi-tenant institutional hierarchy supporting real-world examination workflows.
5) A district-scoped uploader model that reflects the actual organisational structure of examination data collection in Maharashtra state.

XI. LIMITATION AND FUTUREWORK

A. *Current Limitations*

1) In-memory session store: The authSessions Map is lost on server restart and prevents horizontal scaling. A database-backed session store (Supabase or Redis) is required for production multi-instance deployment.
2) Sequential page rasterisation: To manage memory pressure, pages are rasterised sequentially. Parallelisation using worker threads or process pools would reduce OCR latency by 60–70% for multi-page scripts.
3) Mathematical notation: The system cannot evaluate mathematical expressions, chemical equations, or diagrams embedded in answer sheets. STEM subjects with heavy formula content (Calculus, Chemistry) are not well served by the current text-only pipeline.
4) English-only: OCR and scoring logic assume ASCII-compatible English text. Hindi and Marathi medium answer sheets — common in Maharashtra state-affiliated institutions — are not yet supported.
5) Validation dataset size: 16 scripts across three subjects is insufficient for statistically robust accuracy claims. A larger study (200+ scripts, 10+ subjects, multiple institutions) is essential before recommending the system for high-stakes evaluations.
6) No plagiarism detection: The system does not compare student answers against each other, leaving inter-student copying undetected.

B. *Future Directions:*

1) Parallel OCR with worker threads: Isolating each page rasterisation in a Node.js worker_threads worker would enable safe parallel Vision API submission without heap pressure in the main thread.
2) Mathematical expression recognition: Integration of Mathpix Snip or TeX-OCR would extend coverage to STEM subjects by converting handwritten formulae to LaTeX before NLP processing.
3) Multilingual support: Google Cloud Vision supports Devanagari; extending the NLP correction and parsing pipeline to handle Hindi-medium scripts would serve the majority of Maharashtra state-affiliated institutions.
4) Fine-tuned domain embedding model: Fine-tuning a sentence-transformer model on a corpus of annotated (student answer, teacher answer, mark) triples from Indian examination contexts would likely improve scoring accuracy beyond the general-purpose Gemini embeddings.
5) Similarity-based plagiarism detection: Extending compareStructuredAnswers() to run pairwise similarity checks across all students in a cohort would flag suspiciously similar answers for human review.
6) Streaming evaluation: The current bulk evaluation endpoint processes all students sequentially and returns results only on completion. A WebSocket or Server-Sent Events (SSE) streaming interface would provide real-time progress feedback for large cohorts.

XII. CONCLUSION

This paper presented a detailed implementation study of a fully deployed, cloud-native ML-based automated answer sheet evaluation system designed for Indian higher-education institutions. Beginning from a prior conceptual proposal, the team engineered and deployed a production-grade system that integrates Google Cloud Vision OCR, Google Gemini 2.0 Flash for LLM-

based text correction, Gemini Embedding-001 for semantic similarity scoring, Supabase for persistent data management, and a vanilla-JS multi-role web application — all without requiring a build toolchain or external infrastructure beyond a Node.js runtime.

Three technical contributions stand out as novel. First, the three-variant OCR preprocessing strategy — generating original, strong-contrast, and notebook-clean binarised variants of each page and selecting the best candidate by an information-theoretic score — directly addresses the unique OCR challenges of Indian examination notebooks (blue ruled lines, mixed printing and cursive, variable ink density). Second, the five-component lexical similarity metric (Jaccard, bidirectional information-weighted fuzzy token coverage, character tri-gram similarity, and containment score), combined via element-wise maximum, provides robust partial-credit scoring even under substantial OCR noise. Third, the asymmetric minimum-credit heuristic prevents the well-known failure mode of penalising terse but semantically correct answers — a particularly important consideration for short-answer examination formats common in Indian universities.

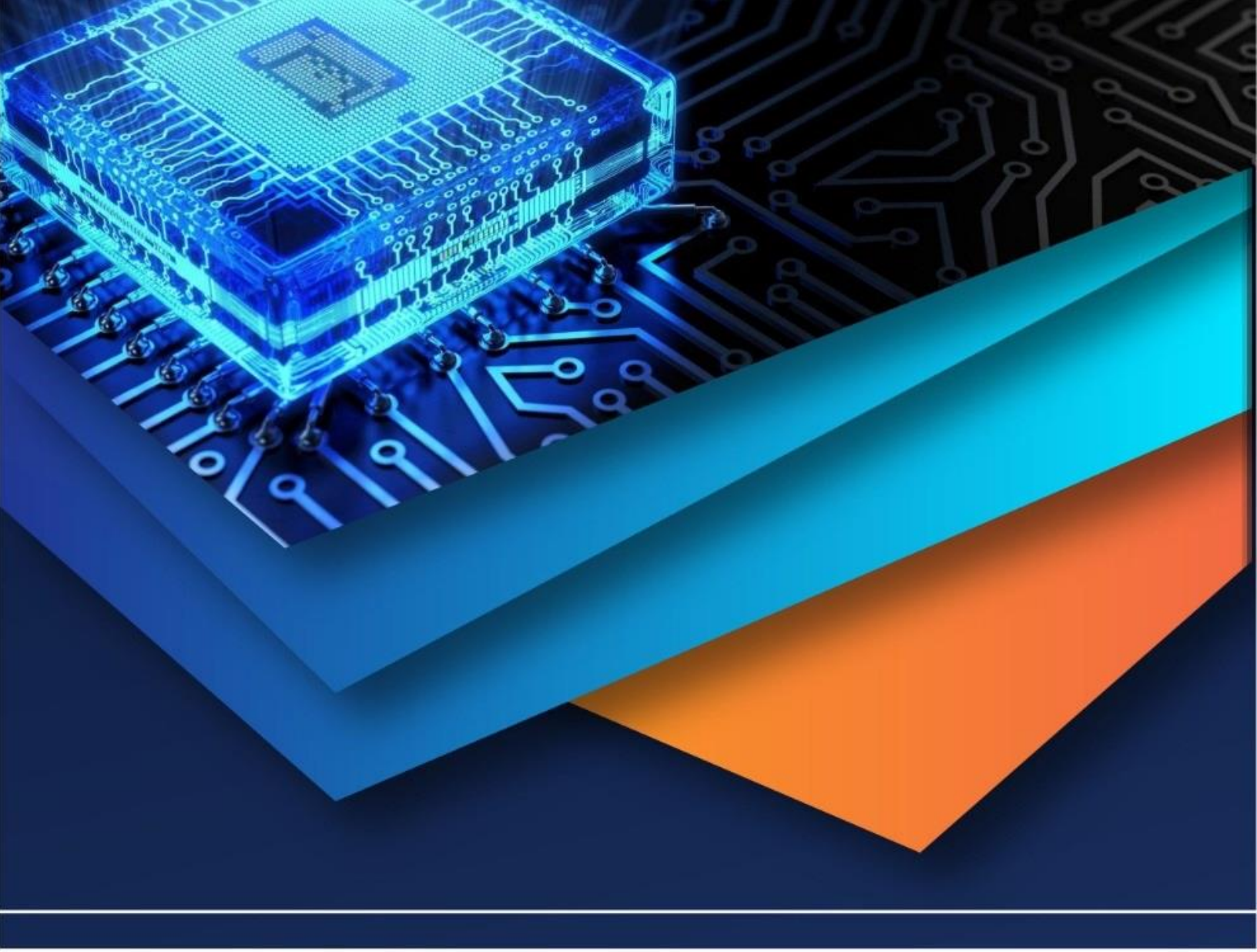
On a validation dataset of 16 scripts, the full system achieved a Pearson correlation of $r = 0.91$ and a within-±1-mark accuracy of 81.2% against teacher-assigned marks — a 20-percentage-point improvement over lexical-only grading and a 9-percentage-point improvement over semantic-only grading. Average end-to-end processing time of ≈ 24 seconds per six-page script enables evaluation of a 50-student cohort in under 22 minutes, delivering a turnaround time several orders of magnitude faster than manual grading.

The implementation documentation provided here — covering OCR pipeline design, question-parser state machine, hybrid scoring mathematics, Gemini API rate-limit management, Supabase storage lifecycle, and multi-tenant RBAC — constitutes, to our knowledge, the most complete published specification of an end-to-end deployed handwritten answer evaluation system. It is intended as a reproducible reference for practitioners in educational technology seeking to build similar systems for their institutional contexts.

REFERENCES

[1] Google Cloud, 'Cloud Vision API — Handwriting Recognition,' Google Cloud Documentation, 2024. [Online]. Available: https://cloud.google.com/vision/docs/handwriting
[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,' in Proc. NAACL-HLT, pp. 4171–4186, 2019.
[3] N. Reimers and I. Gurevych, 'Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks,' in Proc. EMNLP, pp. 3982–3992, 2019.
[4] A. Andhale, R. Chaudhari, P. Derle, O. Jadhav, and K. Ahire, 'ML Based Automated Paper Checking and Evaluation,' Dept. of Information Technology, MET's BKC Institute of Engineering, SPP University, Pune, India, 2024.
[5] S. Bonthu, S. Rama Sree, and P. V. G. D. Prasad Reddy, 'Automated Short Answer Grading Using Deep Learning: A Survey,' in Lecture Notes in Computer Science, vol. 13056, 2021.
[6] M. Leacock and M. Chodorow, 'C-rater: Automated Scoring of Short-Answer Questions,' Computers and the Humanities, vol. 37, no. 4, pp. 389–405, 2003.
[7] M. Sukkarieh and J. Blackmore, 'c-rater: Automatic Content Scoring for Short Constructed Responses,' in Proc. FLAIRS, 2009.
[8] G. Salton and C. Buckley, 'Term-Weighting Approaches in Automatic Text Retrieval,' Information Processing & Management, vol. 24, no. 5, pp. 513–523, 1988.
[9] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, 'Distributed Representations of Words and Phrases and their Compositionality,' in Advances in Neural Information Processing Systems, vol. 26, 2013.
[10] J. Pennington, R. Socher, and C. Manning, 'GloVe: Global Vectors for Word Representation,' in Proc. EMNLP, pp. 1532–1543, 2014.
[11] S. Haller, A. Nisioi, E. Aldea, L. Wolf, and R. Tsarfaty, 'Survey on Automated Short Answer Grading with Deep Learning: From Word Embeddings to Transformers,' arXiv preprint arXiv:2204.03503, 2022.
[12] U. V. Marti and H. Bunke, 'The IAM-Database: An English Sentence Database for Offline Handwriting Recognition,' International Journal on Document Analysis and Recognition, vol. 5, no. 1, pp. 39–46, 2002.
[13] B. Shi, X. Bai, and C. Yao, 'An End-to-End Trainable Neural Network for Image-Based Sequence Recognition and Its Application to Scene Text Recognition,' IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 11, pp. 2298–2304, 2017.
[14] S. Bansal, V. Sharma, and A. Mehta, 'Evaluating Handwritten Answers Using Deep Learning,' in Proc. International Conference on Artificial Intelligence and Education, 2025.
[15] P. A. H. Pawar, 'Automating Handwritten Answer Evaluation: A Deep Learning and OCR Integrated Approach,' Technical Report, 2025.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)