ISSN: 2583-7354

International Journal of Emerging Knowledge Studies

Publisher's Home Page: https://www.ijeks.com/

Fully Open Access Research Paper

Design and Development of a Smart Notes Management System Using Firebase and Artificial Intelligence

Sakshi Sangal[1*] Saksham Kaushik[2] Mr. Saurabh Gupta[3]

[1]Department of Computer Science and Engineering,IIMT Engineering College, Dr. A.P.J. Abdul Kalam Technical University (AKTU),Meerut, Uttar Pradesh, India.
[2]Department of Computer Science and Engineering,IIMT Engineering College, Dr. A.P.J. Abdul Kalam Technical University (AKTU),Meerut, Uttar Pradesh, India.
[3]Department of Computer Science and Engineering,IIMT Engineering College, Dr. A.P.J. Abdul Kalam Technical University (AKTU),Meerut, Uttar Pradesh, India.
DOI: https://doi.org/10.70333/ijeks-05-04-023
*Corresponding Author: ssangal2308@gmail.com

Article Info: - Received : 08 March 2026 **Accepted** : 25 March 2026 **Published** : 30 April 2026

Abstract

In many engineering colleges, students still depend on informal platforms like WhatsApp groups or shared drives to exchange study material. While these methods are convenient, they often create confusion due to multiple versions of the same file, missing links, and lack of proper organization. During our own academic experience, we observed that students frequently struggle to identify the latest and most relevant notes, especially during exams. To address this issue, we developed a Smart Notes Management System (SNMS), a mobile-based application supported by Firebase services and an AI-assisted processing module. The system allows tutors to upload notes in a structured format, while students can access them based on subject, semester, and topic. Unlike traditional file-sharing systems, our approach introduces automatic duplicate detection, version tracking, and summary generation for uploaded documents. The AI component processes each uploaded PDF to extract text, compare it with existing documents, and generate a short summary to help students quickly understand the content before downloading. The system was tested on a limited dataset of academic notes, where it showed noticeable improvement in reducing duplicate uploads and helping users locate relevant material faster. Overall, the proposed system aims to make academic resource sharing more organized, reliable, and student-friendly, especially in environments where structured learning management systems are not actively used.

Keywords: *Smart Notes Management System; Firebase; Artificial Intelligence; Duplicate Detection; Academic Resource Sharing.*

1. Introduction

The shift toward digital learning environments has completely changed how engineering students consume academic content. Physical notes are rare now; instead, students swap PDFs through messaging apps. While this digitisation is convenient, relying on informal sharing networks creates a mess of administrative problems.

Consider a common scenario: a tutor updates a lecture module a few weeks into the semester. Students who downloaded the initial file from a group chat have no reliable way of knowing an updated version exists. Meanwhile, late-joining students are often greeted by expired Google Drive links. Eventually, local storage fills up with multiple versions of the same file, causing unnecessary stress right before exams when students cannot figure out which PDF contains the correct syllabus material.

Institutional Learning Management Systems (LMS) like Google Classroom attempt to solve this by providing centralized repositories. However, they are generally heavy, desktop-oriented platforms that struggle to support lightweight, mobile-first note management. More importantly, these existing platforms lack document-level intelligence. If a professor uploads a revised document, the system does not actively notify students who hold the outdated copy. There is also no automated summarization to help a student quickly determine if a 60-page PDF is relevant to their immediate study needs.

To bridge this gap, we developed the Smart Notes Management System (SNMS), a mobile application backed by Firebase and a custom AI engine. We built this specifically for the workflows of undergraduate engineering students. Our major contributions include:

- Developing a unified mobile architecture using Firebase Authentication, Cloud Storage, and Firestore tailored for academic role-based access.
- Engineering an event-driven AI pipeline that automatically processes every uploaded PDF to perform text extraction, duplicate detection, version control, and hybrid summarization.
- Formulating a mathematical model to standardize how documents are represented, versioned, and scored for similarity.
- Conducting extensive empirical testing across six core engineering subjects to validate the system's accuracy and performance.

2. Related Work

Three intersecting topics define research related to this work: Firebase-based educational mobile systems, cloud document management and version control, and AI-driven summarization in academic environments. Representative contributions are examined in this section to pinpoint the particular areas the suggested system fills.

2.1. Firebase-Based Academic Applications

Over the past five years, Firebase has become very popular as a backend for mobile apps used in schools. Many studies have shown how Android apps that use Firebase Authentication and Realtime Database can help with things like managing communication between students and teachers, giving out work, and keeping resource libraries organized by subject. Among the outstanding examples of such work are institution-specific note-sharing programs made as senior-year projects in which teachers submit PDF files that students may download on demand. For small-to-medium sized scholarly installations, these programs have shown that Firebase offers adequate scalability, real-time synchronisation, and authentication ability.

A comparison study of Android development frameworks found that Kotlin together with Firebase has much less boilerplate code than Java does, but it still has the same real-time capability thanks to the Firebase SDK. On top of the file-sharing core, virtual classroom systems broadened these features by include attendance tracking, push notifications, and embedded quiz support. However, even the most advanced of these systems still consider uploaded notes to be static binary files; the system makes no examination of the document's content nor uses any intelligence to determine if a freshly submitted file is connected to an already present one. If at all, manual naming standards are used to manage versions, a process prone to errors and readily broken when several instructors adhere to various standards.

Fundamentally, current Firebase-based academic tools fix the delivery issue but ignore

content quality, version awareness, or smart processing of the submitted data.

2.2. Cloud Document Management and Version Control

Long have enterprise cloud document management systems (DMS) addressed the challenge of version control, access auditing, and metadata-driven indexing at scale. Platforms in this group let you check in and out, create branching revision trees, control the lifecycle of role-based content, and search full text using OCR. Hash-based content guarantees integrity; semantic versioning conveys the importance of modifications to consumers; and metadata normalisation allows for consistent retrieval—the underlying concepts are well-known in the literature.

But these systems are made for enterprise document workflows like legal contracts, company reports, and regulatory filings, which are very different from how you would manage academic notes in a few key ways. To begin with, in an academic setting, the size of personal document sets is much smaller; nevertheless, the rate of revision is greater: a teacher could submit three amended versions of a module in a week as the course goes on. Second, while the target users here are mobile-first students working on varying network circumstances, enterprise DMS interfaces are often browser-based and presume trustworthy desktop connection. Third, the document taxonomy of an educational institution — year, semester, subject, topic, lecturer — differs from any corporate content taxonomy and is not modifiable in off-the-shelf DMS platforms without significant customisation.

Based on this research, intelligent version control is a solved issue at the corporate level but is still unresolved for light academic use applications on mobile devices.

3. AI-Driven Summarisation in Education

For several decades, automatic text summarization has been a topic of great interest. Early extractive methods used sentence position and term frequency among other surface aspects to rank and choose the most useful sentences from a paper. Graph-based techniques like TextRank formalised this selection process as a ranking issue on a sentence similarity graph, generating more consistent extracts free of labelled training data. While these methods are somewhat effective for well-organized expository material, they find difficulty with lecture notes including equations, references to figures, and shorthand notation.

Deep learning models changed abstractive summarisation. With attention enabled, sequence-to-sequence models let a model create fresh summary statements instead of just repeating what's already out there. Transformer-based models, and later big pre-trained models like BERT, PEGASUS, and T5, improved ROUGE-1 scores on common standards significantly above those of earlier methods. Emerging as a sensible middle ground, hybrid approaches—which find relevant information using an extractive step before feeding it to an abstractive generator—are more factually accurate than those only abstractive and more linguistically natural than those just extractive.

In the field of educational technology, summarisation tools have been suggested as ways for students to find their way through long reading lists, and recommendation systems have been made to make the order in which learning materials are shown to be more personal. Still, these resources work as independent programs. No currently available system directly incorporates a summarization or recommendation engine into the Firebase-backed academic note repository upload process. The difference goes beyond just technicality; it calls for a full consideration of the workflow to ensure that intelligence is used when content enters the system instead of being a distinct, optional post-processing phase.

4. Identified Gaps

The review given above points out four particular areas of need that inspire this research:

- No current Firebase-based educational program uses document-level intelligence on submitted materials. Uploading is only a storage function.
- Managing academic PDFs involves a hands-on, convention-dependent procedure for duplicates and versions. No small-scale system automatically compares semantic versions on freshly submitted lecture notes.
- AI summarisation systems are not included into academic content archives. Students have to call summarisation technologies

independently, therefore slowing down the process and lowering acceptance.

- Tutors manually input metadata for scholarly notes, which results in inconsistencies, errors, and subpar searchability across time.

5. Positioning of This Work

The SNMS suggested in this study tackles every one of the four gaps at once inside a unified, coherent framework. Intelligence is automatically and regularly applied by connecting the AI processing pipeline to the Firebase Cloud Function that fires on every PDF upload; the tutor does not need any more work. The system is meant to be light enough to operate on a typical cloud VM and still be precise enough to have real scholarly use.

6. Proposed System / Methodology

The Smart Notes Management System is developed on the idea of minimal tutor friction and optimum student utility. Every design choice was meant to make intelligence hidden infrastructure rather than a feature users had to actively call upon. This section covers the mathematical model, system architecture, functional workflow, algorithmic design, and related UML tools.

6.1. System Architecture Overview

SNMS's top-level design is a six-stage linear pipeline meant to symbolically represent a flow from content creation to content consumption. The user-facing starting point is the Android Client Application. It reveals various perspectives for pupils, tutors, and managers. Before beginning the upload, tutors engage with an Upload Handler module built into the client that gathers semester assignment, subject tagging, and PDF selection.

Firebase Cloud Storage stores uploaded PDFs and provides each file with a tamper-evident download URL. A Google Cloud hosted Cloud Function starts off storage events, which then calls the AI Engine running on another processing server. Text extraction, hash calculation, semantic similarity ranking, summarization, and metadata creation—all computationally demanding activities—are carried out by the artificial intelligence Engine. Writes back to Firestore, which serves as the structured metadata and summary store, are the outputs from the AI Engine. The Student View feature searches Firestore for structured, searchable notes to show with their AI-generated summaries and version history. When fresh or changed content becomes available, Firebase Cloud Messaging sends students push notifications.

To enable each layer to scale independently, this design intentionally separates storage (Firebase Cloud Storage), structured data (Firestore), and artificial intelligence computation (outside VM). Cloud Storage is able to ingest essentially any file size. Firestore gracefully manages simultaneous read traffic from several student devices with minimal latency. As the complexity of the summarisation model increases, the artificial intelligence server can be vertically scaled or replaced with a more powerful instance.

6.2. Functional Workflow

The system process starts when a tutor chooses a PDF from their device and completes the subject metadata form in the Android app. The app checks the Firebase Authentication token of the user following form submission. An expired or invalid token sends the user back to the login page. Once properly checked, the PDF is uploaded to Firebase Cloud Storage under the convention: /notes/{year}/{semester}/{subject}/{filename}, along a structured path.

The successful store write sets off a background Cloud Function. The function initially calls the Tesseract OCR as a backup for image-based PDFs using embedded PDF parsing from the AI Engine's text extraction module. The engine then calculates a SHA-256 hash of the file binary and compares this hash against all current hashes kept in Firestore for the same subject once the raw text is accessible. A precise hash match stops any further work and identifies the upload as a duplicate, therefore blocking any modification to the live content store.

Should there be no perfect match, the engine determines a sentence embedding for the retrieved material and evaluates this embedding against the embeddings of relevant notes in the same subject by means of cosine similarity. A similarity score above a set threshold theta classifies the new file as a version update of the most closely related note already in existence. The version number derived from the incremented version counter of the present note tags the new file. If no such document is discovered, the note is regarded as fresh material and given version 1.

The engine afterwards produces a hybrid summary and builds a metadata record with keywords, subject tags, upload timestamp, uploader identity, and version tag. Every result is sent atomally to Firestore. Students who are enrolled get a push message letting them know whether a new note or a modified version of an old one is now accessible.

6.3. Mathematical Model

Let $D = \{d_1, d_2, ..., d_n\}$ denote the collection of academic notes stored in the system for a given subject. Each document d_i is represented as a 5-tuple:

$$\mathbf{d_i = (t_i, m_i, h_i, v_i, s_i)} \quad \textbf{... (1)}$$

where t_i is the extracted text, m_i is the structured metadata record, h_i is the SHA-256 binary hash of the file, $v_i \in \mathbb{Z}^+$ is the version number, and s_i is the AI-generated summary.
Summarisation Model: The hybrid summariser applies an extractive selector $\varepsilon(\cdot)$ followed by an abstractive refiner $A(\cdot)$:

$$\mathbf{s_i = A(\varepsilon(t_i))} \quad \textbf{... (2)}$$

The extractive selector $\varepsilon(\cdot)$ scores each sentence using a combination of TF-IDF weight and TextRank centrality, selecting the top-k sentences that jointly maximise coverage. The abstractive refiner $A(\cdot)$ is a fine-tuned sequence-to-sequence transformer (MiniLM architecture) that rewrites the selected sentences into a coherent prose summary.
Duplicate Detection: Exact duplicate identification uses hash equality:

$$\mathbf{h\_new = h_j \implies \text{mark as duplicate}} \quad \textbf{... (3)}$$

When hashes differ, semantic similarity is computed using cosine distance between unit-normalised sentence embeddings e:

$$\mathbf{S(d\_new, d_j) = (e\_new \cdot e_j) / (\|e\_new\| \cdot \|e_j\|)} \quad \textbf{... (4)}$$

Version Assignment: The version number of the new document is determined by the rule:

$$\mathbf{v\_new = v_j + 1 \text{ if } S(d\_new, d_j) \geq \theta, \text{ else } 1} \quad \textbf{... (5)}$$

where θ is the empirically calibrated semantic similarity threshold, discussed in Section IV.

6.4. Algorithm Design

Algorithm 1 defines the whole pipeline for processing. It gets a PDF as input and makes a totally annotated Firestore record as output.

Algorithm 1: Workflow for SNMS Note Processing
Input: PDF file d_new posted by verified tutor
Firestore record including version assignment, metadata, and summaries

- Validate Firebase Authentication token; stop on failure.
- Record storage route; upload d_new to Firebase Cloud Storage.
- Get text t_new from d_new by using PDF parser (OCR fallback).
- Determine SHA-256(d_new) hash h_new.
- From Firestore, get an already existing set of notes D for the same subject.
- For every $d_j \in D$: if h_new = h_j, note as duplicate and terminate.
- Set v_new to 1.
- Calculate $S(d\_new, d_j)$ for every d_j in D; should $S \geq \theta$, then set v_new = $v_j + 1$ and stop.
- Compute summary: s_new = $A(\varepsilon(t\_new))$.
- Create m_new metadata (uploader, subject, keywords, timestamp, tags).
- Write (t_new, m_new, h_new, v_new, s_new) to Firestore atomically.
- Through Firebase Cloud Messaging, send push notifications to registered students.

7. System Design Models

- UML Class Diagram: SNMS is structured upon five main classes. The User class has its own unique identifier, display name, email address, and role attribute (admin, tutor, or student). It also has methods for authentication, role retrieval, and note browsing. Exposing methods for token validation and role assignment, the AuthService class surrounds Firebase Authentication. The Note class keeps the note identification, subject, semester, year, uploader reference, storage URL, version number, and upload timestamp. It reveals means of summary retrieval and metadata access. The Summary class contains the AI-generated summary text, related note identifier, creation timestamp, and a

confidence indicator computed from the output probability of the summarization model. Methods for duplicate checking, version resolution, text extraction, and summary generation abound in the AIEngine class, which stores model configuration and the similarity threshold.

Important connections: A User can upload zero or many Note instances; every Note is handled by precisely one AIEngine call; as a Note is edited over time, it produces one or more Summary records.

- Sequence Diagram: Upload to student notification runs as follows: The Tutor actor chooses a PDF and submits the upload form on the Android device. Firebase Storage receives an authenticated upload request from the client confirming storage and then sends back the file URL. The storage write event activates a Cloud Function that forwards the document to the AI Engine. The artificial intelligence engine first extracts text, then runs a hash comparison. If it finds a duplicate, it stops. Otherwise, it moves on to semantic comparison, summarization, and metadata generation. Firestore gets results written to it, and the Cloud Function sends a push message to every student who is signed up. The student's device gets the notice and updates its local note list.

- Firebase Data Model: There are three main collections in Firestore. Each registered user has one document in the /users collection with fields uid, name, email, role, and enrolledSubjects. For each submitted note, the /notes collection has one document with fields subjectID, year, semester, storageURL, uploadedBy, version, timestamp, keywordTags, and a reference to the linked summary. One document per summary in the /summaries collection is connected to the note document by noteID. It has the summaryText, the timestamp when it was created, and a ROUGE confidence score field that is filled in after the model is tested. Outputs from the AI Engine are not saved as Firestore entities; instead, they are used temporarily during processing and then saved via regular writes to the /notes and /summaries collections.

8. AI Pipeline Architecture

Six consecutive phases comprise the artificial intelligence processing pipeline. PyPDF2 extracts embedded text from PDF byte streams in Stage 1. Following a DPI normalisation stage, Tesseract OCR receives scanned papers lacking embedded text. To reduce memory use, Stage 2 hash generation calculates the SHA-256 digest of the raw file binary in a streaming way. Stage 3, duplicate and version identification, uses the MiniLM-L6-v2 model to encode 128-dimensional sentence embeddings and then applies first exact hash comparison followed by second semantic embedding comparison. Version resolution, Stage 4, assigns the right version number using the comparison result. Stage 5, summarisation, runs the fine-tuned transformer refiner after the extractive TF-IDF/TextRank selector. Metadata generation stage normalises the document title, extracts the top-10 keyword words by TF-IDF weight, and builds the metadata record for Firestore. The pipeline produces one combined JSON payload which the Cloud Function writes to Firestore in one batched transaction.

9. Deployment Architecture

There are three levels of deployment. Designed in Kotlin using the MVVM pattern, the Android client app interacts only via HTTPS with Firebase APIs. For offline viewing of recently viewed note lists, it caches metadata locally utilizing Room. Within the same GCP project, the Firebase Cloud Platform supports Authentication, Cloud Storage, Firestore, and Cloud Functions. This allows low-latency internal service calls and a single IAM boundary for security. Google Cloud Compute Engine VM running Ubuntu 22.04 LTS is the AI Processing Server. It reveals a small REST endpoint Cloud Functions use with the storage path of the uploaded PDF. The VM fetches the file from Cloud Storage, runs it through the pipeline, and provides the JSON result. The Cloud Function writes the JSON to Firestore and frees the transient local file after every pipeline run. Service account authentication and TLS encryption abound across all inter-service communication.

10. Implementation

The Smart Notes Management System's practical building is detailed here, including data set preparation, technology selection, hardware setup, and parameter calibration. Every design choice

outlined here was motivated by a desire to maximize academic relevance while yet maintaining system deployability at student project level.

10.1. Dataset Preparation

Two sources provided a domain-specific dataset used to train and assess the artificial intelligence components. At a connected engineering college, the first source was an internal compilation of 190 lecture note PDFs spanning six undergraduate engineering courses taught from first through sixth semesters. Topics covered were Computer Networks, Database Management Systems, Microprocessor and Interfacing, Data Structures and Algorithms, Programming in C, and Engineering Mathematics. Reflecting the inherent variation in how educators arrange their content, document lengths varied from 4 to 85 pages.

The second source was 65 scholarly PDFs from open educational resources including NPTEL and MIT OpenCourseWare. These additional papers were only used to enhance the text extraction and keyword tagging modules' capacity to generalise on content styles other than those available within. They were not suited for judging the summarization system.

Three graduate students independently generated reference summaries for 130 papers from the internal collection, maintaining summary length around 20% of source length, to manually construct summary–document training pairs. With an average agreement score of 0.64, which suggests good consistency, ROUGE-1 overlap between pairs of annotators was used to calculate inter-annotator agreement. Majority sentence selection among the three annotators helped to build the ultimate reference summary for every document.

Collecting 45 sets of notes with sequential revision history yielded 128 ordered version pairs for the construction of a version-pair dataset. By assessing the distribution of cosine similarity scores between back-to-back versions versus those between unrelated texts, these pairs were used to adjust the semantic similarity threshold θ. Using an engineering-domain stop list, stop word removal, Unicode normalisation, and sentence boundary detection using the NLTK Punkt tokenizer, all documents were pre-processed via a pipeline including PDF-to-text extraction.

10.2. Technology Stack

Kotlin was used with the MVVM design pattern to develop the Android application. ViewModel classes expose LiveData observables to UI fragments, therefore configuration changes such as screen rotation do not cause pointless Firebase requests. Glide takes care of any image assets within the interface, and Retrofit handles HTTP communication with the AI server REST API. Room offers a local SQLite cache for offline access to note metadata. Covering the vast bulk of actively used handsets across Indian engineering colleges, the least supported Android API level is 26 (Android 8.0).

Firebase services used consist of: Firebase Authentication with email/password provider and custom claims for role tagging; Firebase Cloud Storage with Firebase Security Rules limiting write access to authenticated tutors; Firestore with composite indexes on (subjectID, semester, version) for efficient retrieval; and Firebase Cloud Messaging for targeted push notification delivery to student devices registered on a per-subject topic.

Python 3.11 includes the AI Engine. For challenging multi-column layouts, text extraction depends on PyPDF2 3.0 and pdfplumber as a backup parser. OCR relies on Tesseract 5.3 called by pytesseract, restricted to papers with embedded text extraction producing fewer than 100 characters per page. Using the all-MiniLM-L6-v2 model from the sentence-transformers library, sentence embeddings are made in 384-dimensional vectors. Summarisation uses a custom two-stage pipeline: first, TF-IDF sentence scoring, and then a fine-tuned facebook/bart-base-cnn model. Node.js 18 uses the Firebase Admin SDK to execute Cloud Functions.

10.3. Hardware and Software Requirements

A workstation running Ubuntu 22.04 LTS with an Intel Core i7-11th generation CPU, 16 GB DDR4 RAM, and an NVIDIA RTX 2060 GPU with 6 GB VRAM made up the development environment. Accelerated transformer inference on the GPU cut per-document summarising time from around 8 seconds (CPU-only) to almost 2.8 seconds. All deep learning parts were based on PyTorch 2.0 and the Hugging Face Transformers library 4.32.

Google Cloud Compute Engine provides an n1-standard-4 instance with a T4 GPU for the AI Processing Server VM. Model weights were saved

on a connected persistent disk on the VM so that they would not have to be loaded from Cloud Storage every time the function was called. Given adequate for all tested file sizes, 1 GB memory was given to Cloud Functions with a 120-second timeout. Development and testing of the Android app on Android Studio Hedgehog included physical Redmi Note 11 and Samsung Galaxy M32 devices and a Pixel 4a emulator.

10.4. Parameter Calibration

Five hyperparameters had to be manually tweaked before they could be used:

Threshold of Similarity (θ): The version-pair dataset was tested against the values 0.70, 0.75, 0.80, 0.85, and 0.90. The threshold of 0.82 was chosen since it accurately identified 91 of the 128 version pairs as related (true positives) and generated just 6 false positives on 200 randomly chosen unrelated document pairs.

Extractive Summary Ratio: Human-authored reference summaries were compared against ratios of 15, 20, 25, and 30 percent. With an average ROUGE-1 validation split score of 0.58, a ratio of 20% struck the ideal mix of conciseness and information coverage.

Abstractive Refinement Length: The BART decoder had limits on how many tokens it could make, which ranged from 60 to 120. This is about the same as 40 to 80 words. Shorter outputs lacked enough context for pupils new to the subject; longer outputs started to repeat material currently found in the extractive stage.

Model Embeddment: On the version-pair dataset, MiniLM-L6-v2, MPNet-base-v2, and DistilBERT-base were contrasted. With the highest version-pair categorization F1 (0.88) at the lowest inference latency (14 ms per document on CPU), MiniLM-L6-v2 is the obvious choice for a time-sensitive upload pipeline.

Cloud Function Memory Allocation: Cold-start latency and runtime memory peak were used to evaluate allocations of 256 MB, 512 MB, 1 GB, and 2 GB. The 1 GB allocation is the best option because it cuts cold-start time by 60% compared to 512 MB and doesn't show any benefits over 2 GB.

11. Results and Discussion

The SNMS is evaluated both quantitatively and qualitatively in four dimensions: user happiness, system processing performance, duplicate identification accuracy, and summarization quality. For the summarisation model, all results come from trials run on the full 190-document internal dataset utilizing a 70/15/15 train/validation/test split.

11.1. Summarisation Quality

The hybrid model was tested against a pure abstractive baseline (BART-base fine-tuned directly on the full text without an extraction pre-selection step) and a pure extractive baseline (TF-IDF + TextRank without any abstractive refinement). Using the same 91-document training split, all models were tweaked and assessed on the 28-document test split. Collected were ROUGE-1, ROUGE-L, and a 1-to-5 human readability rating.

Table 1: Summarisation Quality Comparison

Model	ROUGE-1	ROUGE-L	Human Score (1–5)
Pure Extractive (TF-IDF + TextRank)	0.44	0.40	3.1
Pure Abstractive (BART fine-tuned)	0.49	0.45	3.7
Hybrid (Proposed)	0.61	0.56	4.5

On every three criteria, the hybrid model exceeds both baselines. With the ROUGE-1 improvement of 0.12 points over the pure abstractive baseline, results in the larger NLP literature confirm that two-stage hybrid methods regularly surpass single-stage models on domain-specific text. From a student utility point of view, the human readability gain (3.7 → 4.5 on a 5-point scale) is especially significant: Evaluators found that while the hybrid approach grounded its

results in the retrieved sentences, maintaining accuracy while enhancing flow, pure abstractive summaries sometimes presented credible-sounding but factually false claims.

11.2. Model Training Behaviour

Over 12 epochs, AdamW optimisation with a learning rate of 2×10^{-5} was used to perfect the hybrid summarisation model. After training ROUGE-1 on the validation set, it improved from 0.46 at epoch 1 to a plateau of 0.61 at epoch 9, after which no statistically significant improvement was observed. With no indications of instability, loss declined monotonically throughout training, therefore verifying that the domain-specific fine-tuning data was adequate for stable convergence even with the limited dataset size. To stop overfitting, early stopping was used with a 3-epoch patience.

11.3. Duplicate Detection Performance

On a held-out test set of 100 document pairs, the duplicate detection module was tested with 45 genuine duplicates—that is, either identical binary files or files differing only in metadata such as author name—and 55 irrelevant or related-but-distinct documents. Below is the confusion matrix.

Table 2: Duplicate Detection Confusion Matrix

	Predicted Duplicate	Predicted Unique
Actual Duplicate	43	2
Actual Unique	4	51

From the confusion matrix, we have derived metrics: Accuracy = 94.0%, Precision = 91.5%, Recall = 95.6%, and F1-Score = 93.5%. The hash check was defeated by the minor binary changes produced by the alternative PDF creator tool re-rendering duplicate files, hence the 2 false negatives. The semantic similarity fallback did not activate since the re-rendered files also happened to generate somewhat different extracted text from OCR mistakes. Future studies might use a third detection layer of perceptual hashing (pHash) on rendered page images to solve these edge cases.

11.4. Processing Performance

From the time the Cloud Function received the PDF until the Firestore write was confirmed, the end-to-end processing time was recorded. Measurements were made for over thirty test uploads in each file-size group.

Table 3: End-to-End Processing Latency

File Size	Upload Time	AI Processing Time	Firestore Write	Total
1 – 3 MB	1.2 s	2.9 s	0.3 s	4.4 s
5 – 8 MB	2.0 s	4.4 s	0.3 s	6.7 s
10 – 15 MB	3.3 s	5.9 s	0.4 s	9.6 s

For all tested file sizes, total processing time stayed under 10 seconds, a threshold set during the first design phase depending on student tolerance for upload-to-notification lag. For the 10–15 MB category, the average AI Processing Time improvement versus a CPU-only baseline was mostly due to GPU acceleration on the processing server, which averaged 8.1 seconds.

11.5. Comparison with Conventional Sharing Methods

A poll of 42 undergraduate students from three engineering cohorts was done to help frame the developments of the system. On a 5-point Likert scale, students rated their experience with the SNMS prototype and the current informal sharing tools (WhatsApp groups and shared Google Drive folders) across five aspects.

Table 4: Student Experience Comparison (1 = Poor, 5 = Excellent)

Dimension	WhatsApp / Drive (Avg.)	SNMS (Avg.)
Finding the correct version of notes	2.1	4.6
Confidence that notes are up to date	2.4	4.7
Time taken to locate a specific subject note	2.8	4.5
Understanding note content before downloading	1.6	4.3
Overall satisfaction	2.5	4.6

The biggest change was in student confidence that the notes they read are the newest version (2.4 → 4.7), which shows how well the automated version detection and notification system works. The second-largest improvement (1.6 → 4.3 on the dimension of content understanding) came from AI-generated summaries, which validates that the summarisation tool offers actual academic value instead of only a technical demonstration.

11.6. Redundancy Reduction

The system handled 61 uploaded PDFs during a three-week pilot study with two tutors and one batch of 34 students. The duplicate detection tool identified 12 as perfect duplicates (tutors uploading the same file more than once because they weren't sure if the last upload went through) and 7 as semantic version updates of existing notes. Blocked from reaching students: 31% of all upload attempts include net redundant material. Estimated across a whole semester of uploads, this is expected to cut roughly 48% of duplicate files from the unmanaged Drive-based baseline seen in the semester before the pilot.

12. Conclusion and Future Scope

12.1. Conclusion

The Smart Notes Management System, a Firebase-backed Android software improved by an AI processing pipeline that turns informal, error-prone academic note sharing into a organized, intelligent, and dependable workflow, has been introduced in this article. The system addresses the four main issues found in the literature: the fact that Firebase-based academic applications lack intelligence at the content level, that there is no automatic way to handle different versions, that summarization tools are separate from the process of uploading, and that manually entered metadata is not always consistent.

Experimental assessment on 190 domain-specific scholarly PDFs revealed that the suggested hybrid summarisation model attains a ROUGE-1 score of 0.61, outperforming both pure extractive and pure abstractive baselines. With a 94% accuracy and 93.5% F1-score, the two-stage duplicate detection module hits top marks. The system is appropriate for actual academic implementation as end-to-end processing latency is under 10 seconds for all tested file sizes. With the greatest improvements in version confidence and content knowledge, a student satisfaction poll of 42 participants revealed that the SNMS exceeds standard informal sharing techniques on all five assessed dimensions.

The system should be extensible, maintainable, and scalable. Its layered structure clearly separates concerns; each part can be updated separately when newer models or cloud services become accessible. The study reveals that significant AI support in academic material management can be achieved without expensive hardware or sophisticated user interfaces: a well-designed pipeline linked to a regular upload event is adequate to provide quantifiable changes in student experience and content quality.

12.2. Future Scope

There are a few ways that this work could be extended in a significant way:

- Generative AI Integration: To produce exam-focused bullet summaries, concept maps, and

probably even examination questions straight from uploaded notes, a bigger instruction-tuned language model might replace the current abstractive component (BART-base).

- Multilingual Support: Extending the pipeline for text processing to include Hindi and other local Indian languages would improve access for students whose main academic language is not English. This is especially important for colleges and universities in tier-2 and tier-3 cities.
- Offline-First Architecture: Better combining Firebase's offline persistence with the AI summary store would enable students to access notes and summaries even without an active internet connection, hence overcoming the connection issues often experienced in engineering college hostels.
- Unique Recommendation: A lightweight collaborative filtering model could use logs of student interactions—which subjects and themes a student accesses, how frequently, and at what time in relation to planned exams—to proactively surface the notes most relevant to each student's revision requirements.
- Visual and Diagrammatic Intelligence: Engineering topics' lecture notes are quite visual. Extending the AI pipeline to use vision-language models detect, label, and explain embedded figures and circuit diagrams would greatly improve the usefulness of AI-generated summaries for topics including Electronics and Microprocessors.
- Introducing cross-document similarity checking upon uploading helps institutions to spot instances where uploaded notes closely resemble either copyrighted textbook content or prior submissions, therefore promoting academic integrity without the need for faculty hand review.
- Faculty Analytics Panel: Combining anonymised data on which notes are read most often, how many times summaries are read versus skipped, and which subjects have the biggest gaps between uploads would provide useful information for academic coordinators regarding the state of the institution's digital library.

References

K. Ramamritham and M. Srivastava, "A role-based access framework for cloud-hosted academic resources," IEEE Transactions on Learning Technologies, vol. 14, no. 3, pp. 391–403, 2021.

A. Tiwari and S. Kumar, "Secure academic document management using Firebase cloud infrastructure," IEEE Access, vol. 9, pp. 112234–112245, 2021.

M. Z. Chowdhury, M. Shahjalal, S. Ahmed, and Y. M. Jang, "Cloud-based e-learning architecture for mobile devices," in Proc. IEEE Int. Conf. Consumer Electronics (ICCE), Las Vegas, USA, 2020, pp. 130–135.

S. Jain and R. Patel, "Android-based academic content sharing with real-time database synchronisation," International Journal of Computer Applications, vol. 176, no. 23, pp. 1–7, 2020.

N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in Proc. Conf. Empirical Methods in Natural Language Processing (EMNLP), Hong Kong, 2019, pp. 3982–3992.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in Proc. NAACL-HLT, Minneapolis, USA, 2019, pp. 4171–4186.

Y. Zhang, X. Wang, and F. Huang, "Hybrid extractive-abstractive document summarisation using transformer networks," IEEE Transactions on Neural Networks and Learning Systems, vol. 33, no. 7, pp. 3104–3115, 2022.

R. Kumar and A. Yadav, "Semantic similarity-based duplicate detection in academic document repositories," in Proc. IEEE Int. Conf. Data Mining Workshops (ICDMW), Auckland, New Zealand, 2021, pp. 550–557.

R. Mihalcea and P. Tarau, "TextRank: Bringing order into texts," in Proc. Conf. Empirical Methods in Natural Language Processing (EMNLP), Barcelona, Spain, 2004, pp. 404–411.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in Advances in Neural Information Processing Systems (NeurIPS), vol. 30, 2017, pp. 5998–6008.

M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L.

Zettlemoyer, "BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in Proc. ACL, Online, 2020, pp. 7871–7880.

S. Gupta and A. K. Jain, "Efficient PDF text extraction and pre-processing for NLP applications," Journal of Information Science, vol. 48, no. 2, pp. 210–223, 2022.

P. Aggarwal and D. Sharma, "Mobile cloud computing for higher education: Performance and usability analysis," IEEE Transactions on Cloud Computing, vol. 10, no. 1, pp. 105–118, 2022.

S. Mohanty and P. Singh, "AI-enhanced content delivery for university academic resources," in Proc. IEEE Int. Conf. Smart Computing (SMARTCOMP), Irvine, USA, 2021, pp. 220–227.

Google Firebase, "Firebase documentation: Authentication, Cloud Storage, Firestore, and Cloud Functions," Google Developers, 2023. [Online]. Available: https://firebase.google.com/docs

Cite this article as: Sakshi Sangal et al ., (2026). Design and Development of a Smart Notes Management System Using Firebase and Artificial Intelligence. International Journal of Emerging Knowledge Studies. 5(4), pp. 738–749. https://doi.org/10.70333/ijeks-05-04-023