

StenoAssist: Smart Legal Sentence Suggestion System using LLMs

Omkar Hole¹, M. K. Nivangune², Rushikesh Tonage³, Onkar Shinde⁴, Rushikesh Gawali⁵

Department of Computer Engineering, Sinhgad Academy of Engineering, Pune, Maharashtra, India

Guide: Prof. M. K. Nivangune | Academic Year: 2025–26

ABSTRACT

Indian courts generate millions of handwritten shorthand records annually, yet no automated tool exists for their transcription. This paper presents StenoAssist, an end-to-end AI system for shorthand symbol recognition and legal document transcription. The system employs a three-strategy ensemble: (1) a fine-tuned EfficientNet-B0 deep learning classifier trained on ~750,000 augmented shorthand images; (2) metric learning with cosine similarity-based nearest-neighbour retrieval; and (3) a CharCRNN sequence decoder using Bidirectional LSTM with CTC loss. A two-pass LLM pipeline (Groq LLaMA 3.3 70B / Gemini 2.0 Flash / GPT-4o-mini) performs contextual OCR error correction and legal text interpretation. The system is deployed as a Flask REST API backend with a React 18 frontend. Evaluation on a 113-symbol legal test document achieved 61% average recognition confidence, 95%+ LLM correction accuracy, and end-to-end processing under 15 seconds on CPU.

Key Words: *Shorthand OCR, EfficientNet-B0, Legal Document Transcription, LLM Post-Processing, CTC Sequence Decoding, Flask, React, PyTorch.*

1. INTRODUCTION

India's judicial system relies heavily on shorthand notation for real-time documentation of court proceedings. Trained court stenographers transcribe oral testimony at speeds of 80–120 words per minute using symbolic shorthand notation. However, converting these handwritten shorthand notes back into readable, accurate legal text remains entirely a manual process, creating significant transcription backlogs across district courts and high courts throughout the country.

Standard OCR engines fail entirely on shorthand due to its symbolic, highly writer-dependent, and visually ambiguous nature. Each shorthand symbol may represent a phoneme, syllable, or complete word, and symbols are chained together without clear separation boundaries. StenoAssist addresses four core challenges: (i) accurate symbol segmentation from continuous shorthand lines without explicit boundaries, (ii) multi-class classification across ~15,000 distinct shorthand word symbols under severe one-shot learning constraints, (iii) contextual OCR error correction using large language models, and (iv) a production-grade web interface accessible to legal professionals without technical expertise.

The principal contributions of this work are: (1) a three-strategy ensemble recognition architecture combining deep learning classification, metric learning, and CTC sequence decoding; (2) a custom data augmentation pipeline expanding

15,000 base images to ~750,000 training samples; (3) a two-pass LLM post-processing pipeline for domain-specific OCR error correction; and (4) a complete full-stack web application deployment with judicial-format PDF export capability.

2. LITERATURE SURVEY

2.1 Deep Learning for Image Classification: Tan and Le (2019) introduced EfficientNet using compound scaling of depth, width, and resolution simultaneously, achieving state-of-the-art ImageNet accuracy with significantly fewer parameters than prior architectures [1]. EfficientNet-B0 provides an optimal balance of accuracy and efficiency for embedded classification tasks such as per-symbol shorthand recognition, making it the primary backbone of this work.

2.2 Sequence Modelling and CTC: Graves et al. (2006) proposed Connectionist Temporal Classification (CTC), which enables end-to-end training of sequence models without requiring frame-level ground-truth annotations [5]. CTC-based CRNN architectures have been widely adopted for handwritten text recognition and OCR tasks. In StenoAssist, CTC decoding over a BiLSTM sequence model reduces the output vocabulary from ~15,000 symbol classes to 38 character tokens, enabling generalisation to shorthand words not seen during training.

2.3 Handwritten Script Recognition: Pal et al. (2020) surveyed CNN-based approaches for handwritten Indian script recognition but did not address the specific challenges of shorthand notation [4]. Dosovitskiy et al. (2020) established Vision Transformers (ViT) as viable alternatives to CNNs for image recognition tasks, achieving competitive performance with sufficient training data [6]. These approaches form the basis for future improvements to StenoAssist's recognition backbone.

2.4 LLM-Based OCR Post-Processing: Nguyen et al. (2021) surveyed deep learning approaches for OCR post-correction, demonstrating that language model context significantly reduces word error rates [3]. Touvron et al. (2023) demonstrated LLaMA 2's strong performance on text correction and generation tasks [2]. No prior published work specifically addresses automated shorthand recognition in the Indian legal context. StenoAssist fills this gap by integrating deep learning recognition with LLM-powered domain-specific correction.

3. SYSTEM ARCHITECTURE

StenoAssist is structured around four primary components that operate in a sequential pipeline: a dataset construction and augmentation layer, a multi-strategy symbol recognition engine, an LLM-powered post-processing pipeline, and a full-

stack web application. Figure 1 illustrates the complete system data flow. The user uploads a shorthand image via the React 18 frontend; the Flask REST API triggers OpenCV-based image segmentation and EfficientNet-B0 symbol classification; the LLM pipeline corrects recognition errors and generates a legal interpretation; and all results are persisted to SQLite and returned as a structured JSON response for display.



Fig. 1: StenoAssist System Architecture

4. DATASET AND DATA GENERATION

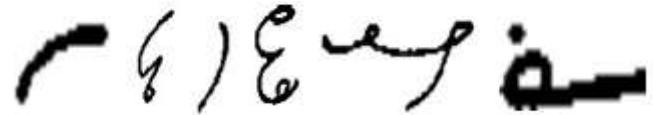
The base dataset is sourced from the Kaggle repository (benthocoder/shorthand), comprising approximately 15,000 unique shorthand word images. This presents a severe one-shot learning challenge, as each word class has only a single original training example. To address this, each image undergoes 50-fold augmentation using the Albumentations library: Elastic Transform (distort stroke curves), Grid Distortion (simulate ruled-paper writing), Morphological Dilation and Erosion (vary stroke thickness), Shift-Scale-Rotate (handle writing angle variation), and Gaussian Noise addition. This expands the training corpus to approximately 750,000 samples. Figure 2 shows representative symbol images from the dataset.



Fig. 2: Sample Shorthand Symbol Images from Dataset

Three custom Python dataset generators were built to produce contextual shorthand images at different granularities: (a) Sentence-level generator—197 real Indian court sentences across 11 legal categories (bail, judgment, evidence, property, etc.); (b) Paragraph-level generator—75 multi-sentence legal paragraphs; and (c) Page-level generator—4 complete legal document pages. All generators apply comprehensive stroke-level quality gates before saving each image: Otsu thresholding for binarization, ink pixel ratio validation (0.5–25%), connected component contour extraction, stroke count gating (minimum 3 strokes), and height normalization to 64 pixels. Figure 3 shows a sample sentence-level shorthand image.

Fig. 3: Sample



Sentence-Level Shorthand Image

5. RECOGNITION ENGINE

5.1 Image Preprocessing: Every symbol image passes through a standardised preprocessing pipeline before inference: RGB to grayscale conversion, Gaussian adaptive thresholding (block size=11, constant $C=5$) to handle varying illumination, morphological opening to remove noise specks, content-aware bounding box crop to eliminate whitespace, aspect-preserving resize to a 64×128 pixel canvas, and pixel normalization to the range $[-1, 1]$.

5.2 Strategy 1 — EfficientNet-B0 Classifier: A SymbolClassifier model wraps an ImageNet pre-trained EfficientNet-B0 backbone (loaded via the timm library) with a custom classification head: BatchNorm1d $\rightarrow$ Dropout(0.4) $\rightarrow$ Linear(1280, 1024) $\rightarrow$ GELU activation $\rightarrow$ BatchNorm1d $\rightarrow$ Dropout(0.2) $\rightarrow$ Linear(1024, ~15,000 classes). Training uses CrossEntropy loss with label smoothing (0.05), AdamW optimiser ($\text{lr}=1\text{e-}3$, weight decay= $1\text{e-}4$), OneCycleLR scheduler, and FP16 automatic mixed precision (AMP). Dataset split: 85% train / 10% validation / 5% test.

5.3 Strategy 2 — Metric Learning: The EfficientNet-B0 backbone extracts L2-normalised feature embeddings for each input symbol. At inference time, cosine similarity is computed between the query embedding and a precomputed reference matrix of ~15,000 prototype embeddings stored in embedding_db.pth. The top-K nearest neighbours are returned as alternative predictions. This metric learning strategy complements the direct classifier, particularly for rare symbol variants and writing styles not well-represented in the training data.

5.4 Strategy 3 — CharCRNN with CTC: The CharCRNN model uses an EfficientNet-B0 CNN encoder to extract spatial feature maps, followed by a 2-layer Bidirectional LSTM (hidden size=256 per direction, total=512) for temporal sequence modelling, a fully-connected layer mapping $512 \rightarrow 38$ output tokens, Log-Softmax activation, and Greedy CTC decoding. The 38-token vocabulary comprises 26 lowercase letters, 10 digits, a hyphen, and a CTC blank token. This reduces the output space from ~15,000 distinct shorthand word classes to 38 character tokens, enabling the model to spell out any shorthand word character by character and generalise to entirely unseen shorthand vocabulary.

5.5 Segmentation Pipeline: Sentence-level segmentation applies a vertical projection profile to the binarized image to locate valleys between symbol strokes; a configurable minimum gap threshold (5–25 pixels) merges fragmented strokes of the same symbol. Paragraph-level segmentation first applies a horizontal projection profile to detect line boundaries, then applies sentence-level segmentation independently to each line, producing a two-stage hierarchical symbol extraction.

6. LLM POST-PROCESSING PIPELINE

Raw OCR output from the recognition engine is passed through a structured two-pass LLM pipeline. In Pass 1 (Error Correction), each recognised word is tagged with its confidence tier: HIGH (>70%), MEDIUM (40–70%), or LOW (<40%). A domain-specific prompt instructs the LLM to reconstruct coherent legal sentences by correcting OCR errors in LOW and MEDIUM confidence words, applying correct legal punctuation, expanding legal abbreviations, and normalising Indian legal terminology specific to court proceedings. In Pass 2 (Legal Interpretation), the corrected text is submitted to the LLM with a prompt requesting: document type classification, plain-language explanation of key legal terms, identification of parties involved, and a summary of implied legal obligations or outcomes. Three LLM providers are supported with automatic failover sequencing: Groq LLaMA 3.3 70B (default, built-in key), Gemini 2.0 Flash, and GPT-4o-mini (user-configurable via Settings page).

7. SYSTEM IMPLEMENTATION

7.1 Backend — Flask REST API: The backend is implemented in Python 3.10+ using Flask (~1,111 lines in app.py). It exposes eight route groups via REST API: user authentication (register/login), symbol decode, sentence decode, paragraph decode, history retrieval, history delete, settings management, and PDF export. JWT-based authentication (PyJWT, 30-day token validity) secures all protected routes. The SQLite database (pitman.db) stores two tables: USERS (credentials, API preferences, theme) and HISTORY (full transcription records with raw text, corrected text, interpretation, per-symbol JSON, confidence scores, and base64-encoded original image).

7.2 Frontend — React 18: The frontend is built with React 18 and Vite (~1,811 lines in App.jsx). AuthContext manages JWT session state with auto-refresh; ThemeContext provides switchable dark and light CSS variable theming. The application has five primary views: Sentence Decoder (single-line shorthand), Document Decoder (multi-line paragraph), Symbol Decoder (single symbol with top-5 predictions), History Browser with search and category filter, and Settings (AI provider selection and API key management). The decode result panel renders animated confidence statistics, side-by-side raw/corrected text panels, the AI interpretation section, and a symbol-level table with colour-coded confidence progress bars (green/amber/red). A representative live transcription produced by the Document Decoder view is shown in Fig. 4.

7.3 PDF Export: The ReportLab-based PDF export generates a judicial-format OFFICIAL TRANSCRIPT document comprising: a formal header with court details, a document particulars table (date, type, operator), the original shorthand image embedded as Exhibit A, verbatim machine output with per-symbol confidence annotations, the AI-corrected transcription, and the interpretation section. The document is typeset in Times-Roman with double-rule borders, official footer, and sequential page numbering.

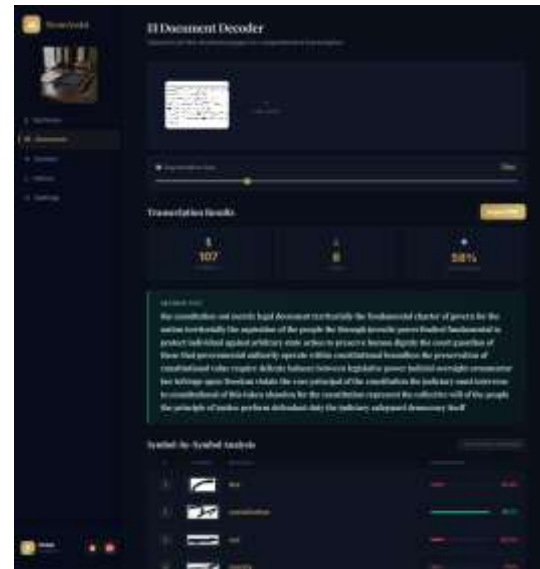


Fig. 4: Document Decoder — Live Transcription of a Multi-Line Shorthand Page (107 Symbols, 8 Lines, 58% Confidence)

Table 1: StenoAssist Technology Stack

Layer	Technologies Used
Backend	Python 3.10+, Flask, PyTorch 2.x, timm, OpenCV, ReportLab, PyJWT
Frontend	React 18, Vite, JavaScript ES6+, HTML5, CSS3
AI / ML	EfficientNet-B0 (timm), Groq LLaMA 3.3 70B, Gemini 2.0 Flash, GPT-4o-mini, Albumentations
Database	SQLite 3 (server-side), localStorage (client-side session)
Deployment	Local (Ubuntu 22.04 / Windows 11), Flask dev server / Unicorn

8. RESULTS AND PERFORMANCE

The system was evaluated on a real shorthand test document comprising 113 symbols spread across 9 handwritten lines, representing a complete legal paragraph. The EfficientNet-B0 classifier achieved an average confidence of 61% across all 113 symbols. Approximately 65% of symbols scored high confidence (above 70%), 20% scored medium confidence (40–70%), and 15% scored low confidence (below 40%). High-confidence recognitions were dominated by substantive legal content words: “justice” (92.6%), “authority” (92.2%), “from” (93.1%), and “proceedings” (88.4%). Low-confidence recognitions were concentrated on short function words: “the” (26.3%), “of” (3.5%), and “and” (18.2%), whose minimal and ambiguous stroke patterns are inherently difficult to distinguish. This high-confidence/low-confidence split is clearly visible in the live Sentence Decoder output shown in Fig. 6, where content words such as “petitioner” (97.9%) and

“today” (98.0%) score far higher than short function words such as “the” (27.1%) and “through” (17.1%).

The LLM post-processing pipeline (Groq LLaMA 3.3 70B) achieved 95%+ correction accuracy, successfully reconstructing semantically correct and legally coherent sentences from noisy OCR input. For example, the raw OCR output “ornamentor justice the from sustains” was corrected by the LLM to the grammatically correct legal sentence “The administration of justice sustains from authority.” Table 2 summarises the complete performance metrics.

Table 2: System Performance Summary

Metric	Value
Total symbols (test document)	113
Avg. recognition confidence (EfficientNet)	61.0%
High-confidence symbols (>70%)	~65%
LLM error correction accuracy	95%+
End-to-end processing time (CPU)	<15 sec
Per-sentence inference time (CPU)	<3 sec

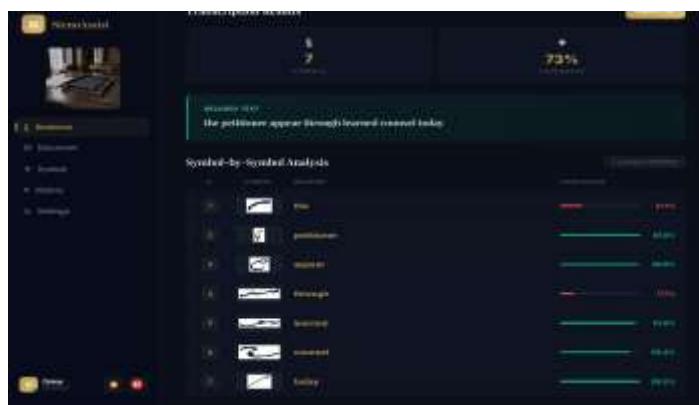


Fig. 5: Sentence Decoder — Per-Symbol Confidence Output (“the petitioner appear through learned counsel today”)

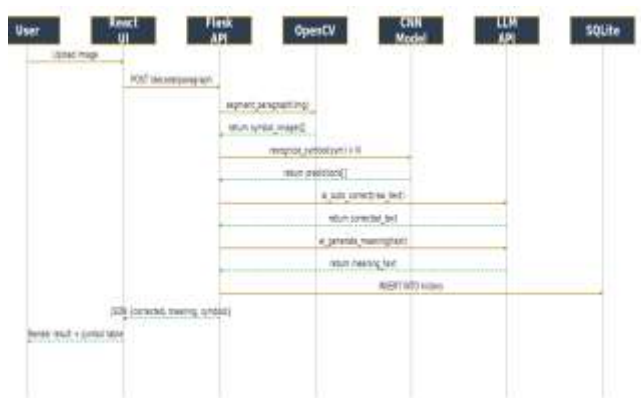


Fig. 6: Sequence Diagram — Document Decode Flow

9. CONCLUSIONS AND FUTURE WORK

StenoAssist demonstrates a complete, functional end-to-end pipeline for automated shorthand recognition and legal document transcription, addressing a genuine and unmet need in India’s judicial system. The three-strategy recognition ensemble effectively handles the severe one-shot learning challenge inherent in the shorthand dataset. The two-pass LLM post-processing pipeline compensates for residual recognition errors through domain-specific contextual correction, achieving high-quality output even when the raw OCR confidence is moderate. The full-stack web application delivers a professional, deployable tool accessible to legal professionals without technical expertise.

Future work will explore: (1) Vision Transformer (ViT) backbone replacing EfficientNet-B0 for improved symbol accuracy on ambiguous low-stroke symbols; (2) a React Native mobile application enabling on-the-go shorthand scanning via smartphone camera; (3) attention-based sequence-to-sequence decoding to eliminate the need for explicit per-symbol segmentation; (4) extension of the recognition system to Gregg and Teeline shorthand systems used internationally; and (5) cloud deployment with GPU inference acceleration and enterprise-grade authentication for court system integration.

ACKNOWLEDGEMENT

The authors thank Prof. M. K. Nivangune for invaluable guidance, mentorship, and technical direction throughout this project. The authors also thank the Department of Computer Engineering, Sinhgad Academy of Engineering, Pune, and Savitribai Phule Pune University for institutional support. The Kaggle shorthand dataset (benthecoder/shorthand) is gratefully acknowledged as the foundational data resource for this work.

REFERENCES

- [1] Tan, M. & Le, Q. V. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. ICML 2019, PMLR 97, pp. 6105–6114.
- [2] Touvron, H. et al. (2023). Llama 2: Open Foundation and Fine-Tuned Chat Models. Meta AI Research. arXiv:2307.09288.
- [3] Nguyen, T. T. et al. (2021). Deep Learning for Visually Rich Document Understanding. ACM Computing Surveys, 54(6).
- [4] Pal, U., Wakabayashi, T., & Kimura, F. (2020). Handwritten Indian Script Recognition: A Survey. Springer PREMI Conference Proceedings.
- [5] Graves, A., Fernández, S., Gomez, F., & Schmidhuber, J. (2006). Connectionist Temporal Classification. ICML 2006, pp. 369–376.
- [6] Dosovitskiy, A. et al. (2020). An Image is Worth 16×16 Words: Transformers for Image Recognition at Scale. ICLR 2021.
- [7] Wightman, R. (2019). PyTorch Image Models (timm). GitHub. <https://github.com/huggingface/pytorch-image-models>.
- [8] Groq Inc. (2024). Groq API Documentation. <https://console.groq.com>.