INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD)
© 2026 IJNRD | Volume 11, Issue 5, May 2026 | ISSN: 2456-4184 | IJNRD.ORG

AUTOMATED TEXT TO HAND WRITING GENERATION USING MACHINE LEARNING TECHNIQUES.

1st Sudharshan.S.S

Department Of Advanced Computing And Analytics

Vels Institute Of Science, Technology & Advanced Studies

Chennai, Tamil Nadu

sudhasrhan.s.sbca@Gmail.Com

2nd Dr.R.Balamurugan

Department Of Advanced Computing And Analytics

Vels Institute Of Science, Technology & Advanced Studies

Chennai, Tamil Nadu

bmurugan.scs@vistas.ac.in

Abstract— ***Writing long notes or homework by hand eats up hours, yet most learners find it tiresome instead of satisfying. Built to lighten that load comes a digital helper. A website transforms regular keyboard text into pictures styled like human pen work. Behind the scenes, software studied thousands of handwritten examples before linking to an online page. Type words there, choose how they appear, then save what you made - just like paper filled with ink strokes. Python runs the back end alongside TensorFlow and Flask, yet the front end sticks to basic HTML, CSS, and JavaScript. Character by character, the model learns from 62 types of English symbols - lowercase, uppercase, numbers included. Each symbol gets turned into a tiny image before they're joined into a single result. From there, the final image may become either a standard picture or a PDF document. For people needing hand-drawn text often, without hours to write it all down, this cuts effort dramatically.***

Keywords—*text to handwriting, handwriting generation, LSTM, TensorFlow, Flask, image synthesis, deep learning,*

I. INTRODUCTION

Writing something by hand sometimes takes forever, especially when pages are needed. One task might demand two or even three hours of constant scribbling, which piles up as weeks go by. Yet printed words on paper lack the personal touch of real pen strokes, making digital work feel off in some cases. Because of this, just using a keyboard won't fix everything. The slow grind of handwriting stays unavoidable now and then.

Starts with shapes meant to mimic pen strokes, most software leans on fixed font designs. Try them out, notice how they seem real until you take time to study - each character repeats itself without change. Actual hand-drawn letters shift subtly throughout a line, even within a single word. That little difference in form, like an earlier 'a' versus one written later, adds life. Imperfect rhythm gives it authenticity.

A different approach kicks in here - using a machine learning system shaped by actual photos of hand-drawn symbols. Rather than copying shapes from fixed digital fonts, it builds every letter out of tiny pixels, guided by what it picked up while studying examples. Each symbol gets crafted separately, later lined up next to others, fitting together like pieces to build full lines of text.

Running entirely in your browser means no setup needed. Into a box on screen goes whatever words someone wants to write by hand. Changing how it looks happens next - things such as pen shade or letter height get picked. After pressing go, out comes a picture made to mimic real script. That snapshot waits ready for saving right away if wanted. Or else every page stacks neatly into one document saved as PDF at once.

II. LITERATURE SURVEY

Before settling on an approach for this project, a review of related work was carried out to understand what had already been tried and where the gaps were.

Study 1: Font-Based Handwriting Tools

The most common approach seen in existing tools is to use a specially designed font that looks like handwriting. These fonts are available in a large number of styles and are very easy to apply to any piece of text. The problem is that fonts are static. Every occurrence of a letter uses exactly the same glyph, which gives the final output a mechanical quality. Because there is absolutely no variation between the same characters on the same page, a careful reader is able to quickly identify that the text was not written by hand.

Study 2: LSTM Networks and Handwriting Synthesis

In 2013, Alex Graves worked on an important study. His studies displayed how Long Short-Term Memory networks, a type of recurrent neural net, could create exact handwriting using trends in pen movement data. These nets work well because writing happens over time and their structure preserves historical details while influencing future results.

Study 3: GAN-Based Approaches

Compared to earlier, easier techniques, the written lines are sharper as the result of that back-and-forth tug. However, these networks cause problems because they require machines that are much more powerful than the typical college notebook can

provide, are picky during training, and require far more examples than we had here.

Research Gap

Considering every thing, it is obvious that there is a gap in the tools at hand. On the one hand, font-based tools are simple to use but show up phony. However, there are GAN systems at the research level which produce great results but are out of reach for the majority of people. Something in the middle is lacking: a lightweight model that learns from actual handwriting data, going beyond fonts, and is presented in an easy-to-use web interface that doesn't require technical expertise. This project seeks to occupy that area.

III. PROPOSED METHODOLOGY

The project was built in phases, beginning with the data and progressing through model training to the web application. Below are descriptions of each step.

A. Data Collection

The English Handwritten Characters Dataset on Kaggle provided the training data. The 3,410 greyscale images in this dataset are divided into 62 character classes, each containing 55 images. All 26 capital and lowercase letters, as well as the ten digits from 0 to 9, are covered in the classes. Every image path has been assigned to its matching character label in a CSV file. The dataset was selected because it included all the characters required to render standard English text, was reasonably clean, and was accessible to the public.

B. Data Preprocessing

To eliminate any color information that could confuse the model, each image in the dataset was opened using the Pillow library and converted to greyscale. In order to ensure that all inputs to the model had the same dimensions, each image was then resized to 500 by 500 pixels. Training became more stable after the pixel values were normalized by dividing them by 255, bringing them into the range of 0 to 1. Each of the 62 character classes was given a distinct integer using a Keras character-level tokenizer. The entire image dataset was 3410 × 500 × 500 after preprocessing.

C. Feature Engineering

The integer token for each character serves as the input for each prediction since the model operates one character at a time. In order to create visible spaces between words, space characters in the user's text are handled independently by creating a plain white image with pixel values set to 249. This image neatly slots into the combined output.

D. Model Architecture

Built using TensorFlow along with Keras, this model uses a setup of three layers.

E. Training and Inference

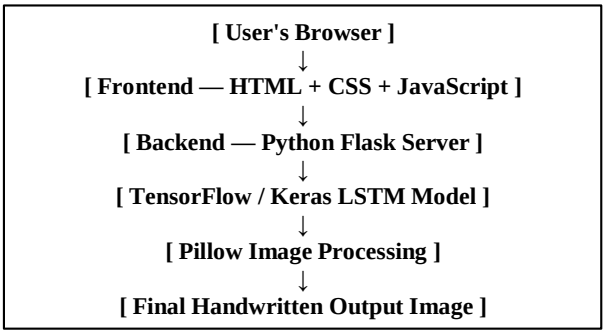
Training ran for 300
epochs, each using 64 samples at once. At first,the team tried learning from whole words, yet progress stalled - loss refused to drop. Only when shifting to one letter at a time did everything start moving smoothly. When making predictions now, the system breaks down the inputinto separate letters before processing. A single character gets turn ed into pixels by the model, forming a 500 ×500 frame. From that, every image shrinks down - cropped right at the middle - to fit 250 ×
500. Side by side they line up, these trimmed pieces, building the f ull result in sequence.

F. Web Interface

One HTML page forms the front, shaped by CSS, brought to life through JavaScript. On the left sits a digital A4 space - users type there. To its right, controls appear in a sidebar layout. Clicking generate triggers html2canvas, turning the designed view into a canvas snapshot ready for download. For longer outputs, jsPDF stitches several such images together inside a single PDF file [7]. Over on the server, Flask collects what was typed, pushes it through a pre-trained network, then sends back the resulting picture.

IV. ARCHITECTURE DIAGRAM

The diagram below shows how the different parts of the system connect to each other.



V. VARIOUS PHASES AND METHODOLOGIES

Phase 1: Data Acquisition and Exploration

To make sure things were balanced, someone counted how many tim es each of the 62 labels showed up - turned out every one had exactly 55 entries. After that came a scan for gaps or repeated rows; nothing popped up there either.

Phase 2: Image Preprocessing Pipeline

In this part where code was written inside a Jupyter
Notebook to handle data prep. Step by step, results were checked just to be sure everything worked right. Once images went through, their final form showed up as (3410,
500,500). That shape meant things had gone correctly so far. On anot her path, tokens got sorted into exactly 62 different labels same count we were supposed to get.

Phase 3: Model Design and Training

A single word came in, an entire word image went out - that was
the original setup.
Training dragged on forever yet losses barely budged. Turns out, gen erating full
word images demanded more examples than we had. Zooming into c haracters instead made everything click. Progress finally moved smo othly after that shift.

Phase 4: Inference and Image Merging

After training finished and the file stored, a prediction setup formed around get_out_image. From that point on, it took care of breaking up sentences, making guesses, organizing where pictures go, also triggering the merge process. When tested with small phrases such as "Apple" or "Cat and Dog", things looked promising

VI. INPUT

A. Training Input

During the training phase, the model received individual character samples from the dataset. Each input was the integer token for a single

character, shaped as (1, 1), and the corresponding target was the normalised pixel array of the handwritten character image, flattened to a vector of 250,000 values.

B. User Input During Inference

When the application is running and a user interacts with it, the main input is a typed text string. The user can also configure a number of settings before generating the output:

- Handwriting Font: choice of Homemade Apple, Caveat, Liu Jian Mao Cao, or an uploaded .ttf or .otf file
- Ink Colour: blue (#000f55), black, or red (#ba3807)
- Font Size: entered in points, default 10pt
- Page Size: A4
- Page Effects: shadows, scanner simulation, or no effect
- Resolution: five levels from 0.8x up to 4x
- Spacing: vertical text position, word spacing, and letter spacing
- Margin and Line toggles: paper margin on/off, ruled lines on/off
- Optional background: a custom paper texture uploaded as JPG or PNG

VII. PSEUDOCODE AND IMPLEMENTATION

A. Pseudocode

```
BEGIN
-- TRAINING --
Load chardata/english.csv
FOR each image row in CSV:
  Open image with Pillow
  Convert to greyscale
  Resize to 500 x 500 pixels
  Convert to NumPy array
image_data = np.array(output) / 255.0
Fit Keras Tokenizer on label column
text_sequences = tokenizer
  .texts_to_sequences(labels)
Build model:
  Input -> Embedding(63, 50)
  -> LSTM(512 units)
  -> Dense(250000, activation=relu)
Compile: Adam, loss=mean_squared_error
model.fit(sequences, image_data,
  epochs=300, batch_size=64)
model.save('my_model.keras')
-- INFERENCE --
model = load_model('my_model.keras')
FUNCTION get_out_image(text):
  FOR each character in text:
    IF space: append white_image_249
    ELSE: pred = model.predict(seq)
          .reshape(500, 500)
  FOR each saved PNG:
    cropped = center_crop(img, 250x500)
  merged = merge_images(images)
-- WEB LAYER --
ROUTE POST /generate-handwriting:
  text = request.json['text']
  Call get_out_image(text)
  Return base64 PNG to frontend
Frontend:
  POST to Flask via fetch()
  Inject image into output div
  Offer PNG or PDF download (jsPDF)
END
```

B. Implementation Notes

The most important decision in the implementation was moving from word-level to character-level prediction. The word-level model was trained first but never converged. At character level, predictions made things far easier to handle. Instead of coping with many thousands, just 62 conversions were needed. Each one links an individual number to a picture sized 500 by 500.

Splitting JavaScript into pieces helped more as things got bigger on the front end. Instead of mixing everything, draw.mjs took shape on its own before sliding into place as a test part. Grabbing images felt smooth thanks to html2canvas, just so long as the picture finished drawing first.

VIII. OUTPUT

A. Per-Character Generated Images

A single grayscale PNG comes out for every letter typed in. These pictures measure exactly 500 pixels wide by 500 tall, shaped by an LSTM network's decisions. Though the same character appears more than once, slight shifts in shading appear across copies. That subtle shift - never quite repeating - is why it feels hand-drawn instead of stamped like a digital font.

B. Merged Handwriting Image

A single stretched picture forms by placing each letter's frame beside the next, sized exactly at 250 by 500 pixels. White rectangles stand where spaces go, breaking up the word clusters clearly. What shows on screen comes from linking these cutouts side by side. Clicking save grabs that combined version shown below the typing area.

C. Styled Page Download

A different look comes out with the main result, shaped by your choices. Your words appear on a screen-sized sheet, drawn in the handwriting style you picked - down to the ink shade and how far apart the lines sit. That image gets saved through html2canvas, turning into a downloadable PNG. Ruled lines might show, margins hold their place, and any paper-style touches stay visible in the final picture.

D. PDF Export

Clicking the Download All Images as PDF
button works when multiple pages are made. One after another,each c anvas gets pulled by jsPDF into one combined PDF. What results is u seful full tasks saved, not fragments.The whole thing ends up feeling less like pieces, more like finished work.

E. Test Outputs Observed

A single word Apple became a line of five uneven letters, hand- drawn and attached sideways into one strip. Each character shifted slightly, as if written at different moments by the same unsteady hand. One leans left, another thickens at the base, while a third tapers like it ran out of ink. These differences weren't smoothed away; they stayed raw, part of the texture. The whole thing stretches flat across, connected but never matching A cat shows up beside a dog in the output, forming one stretched picture. Three words appear split by obvious gaps of empty area. Each term stays visible, divided clearly across the frame Uppercase and lowercase letters showed up exactly where they should after entering Test Input Sample. The system handled varied capitalization without shifting any character placement.

IX. RESULT AND DISCUSSION

A few different inputs were used to test how the system responds, with output inspected by eye each time. Besides that, every clic kable part ofthe website was tried out to see if things worked as expe cted.

A. Model Performance

Starting off smooth, the character-level LSTM trained without serious hiccups. Every one of the 300 epochs showed a steady drop in loss. Though only 55 examples were used for each category, that

number held up well enough. Recognizable handwritten forms emerged for all 62 groups. Letters built from many lines - say, certain capital letters - ended up fuzzier around the edges. Meanwhile, leaner symbols such as the numeral 1 or small l kept their crispness.

B. Output Quality

Most of the time, short and medium handwritten lines came out clean after merging. Pixel details looked natural, something you do not see with standard fonts. When words got longer, small shifts up or down sometimes happened where letters met. This appeared more often when big and small letters were together in one word. Training only on single characters meant the system did not learn how neighbors line up across a word. Alignment gaps followed from that.

C. Customisation

All the customisation options worked as intended during testing. Switching ink colour from blue to black or red produced a clear change in the output. Increasing the resolution setting made the final image crisper at the cost of a slightly longer generation time. The ability to upload a custom handwriting font was tested with a third-party .ttf file and worked without issues. The scanner effect gave the output a convincingly aged look.

D. Known Limitations

- The ML model only covers 62 alphanumeric character classes and cannot handle punctuation marks, special characters, or any non-English script
- Characters are merged as independent image blocks, so there is no cursive joining between letters even when a cursive-looking font is selected
- The model was trained on isolated characters, meaning it has no understanding of how one letter should connect visually to the next
- Animated pen stroke output, where the writing appears to be drawn in real time, is not currently supported.

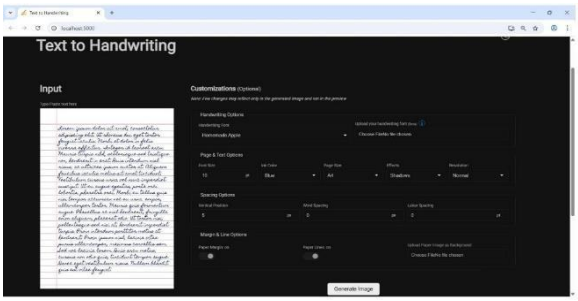
E. Comparison with Existing Approaches

TABLE I COMPARISON OF HANDWRITING GENERATION APPROACHES

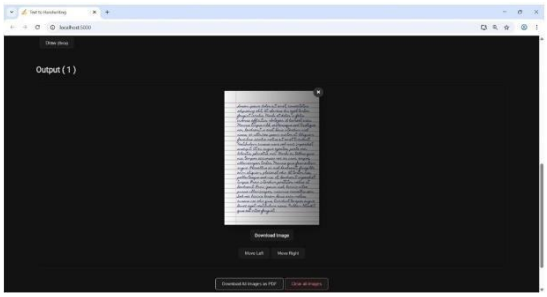
Feature	Font-Based	GAN-Based	This Project
Looks like real handwriting	Low	High	Moderate to High
User customisation options	Limited	Limited	Extensive
Compute resources needed	Very Low	Very High	Low
Accessible via web browser	Yes	Rarely	Yes
PDF export	Sometimes	Rarely	Yes
Easy to deploy	Yes	No	Yes

X. SCREENSHOTS, CHARTS, AND GRAPHS

The figures below show the web interface and sample outputs generated during testing.



[Fig. 2: Main Input Interface Screenshot]



[Fig. 3: ML Output — Input "Test Input Sample"]

The Jupyter Notebook training session also produced the following outputs that were useful for verifying the system worked correctly:

- A training loss plot showing the loss curve falling steadily over 300 epochs, confirming that the character-level model was learning
- A greyscale plot of a sample character image (the digit 3) confirming that images were loading and normalising correctly
- Visual output from the get_out_image function for three test strings, showing that the character merging logic produced coherent handwritten words

XI. CONCLUSION

This research introduced a program capable of turning typed text into handwritten-looking pictures using a machine mind shaped by practice. Why bother? Handwriting still takes up too much time these days. Alternatives tend to look fake, need heavy-duty machines, or demand deep tech skills just to run properly.

One letter at a time - that became the new start instead of full words trained on handwritten samples. Halfway forward, shrinking down to single characters shifted how things unfolded. Progress grew smoother once it switched paths, no longer freezing up like before. As tests ran, each of the 62 symbols, digits, and letters slowly took shape into something clear. Viewers saw those outputs appear right inside their web browsers while the demonstration played out.

With this tool, picking your own ink shade or typeface isn't just possible - it shapes how personal the final page feels. Instead of getting something stamped out the same for everyone, you adjust spacing, layout, even paper texture, so it mirrors individual handwriting styles. Control like that shifts the outcome from robotic to familiar, almost human at a glance.

Possibly next steps come into view when thinking about where things might head. With exposure to broader examples during training, results tend to get clearer and hold steady.

REFERENCES

[1] A. Graves, "Generating Sequences With Recurrent Neural Networks," arXiv preprint arXiv:1308.0850, Aug. 2013.

B. Davis, N. Morse, B. A. Davison, B. Bhanu, and J. Mayfield, "Text and Style Conditioned GAN for Generation of Offline Handwriting Lines," in Proc. British Machine Vision Conference(BMVC), 2020.

[2] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[3] M. Abadi et al., "TensorFlow: A System for Large-Scale Machine Learning," in Proc. 12th USENIX Symposium on OSDI, 2016, pp. 265–283.

[4] D. Dhruvil, "English Handwritten Characters Dataset," Kaggle, 2021. [Online]. Available: https://www.kaggle.com/datasets/dhruvildave/english-handwritten-characters-dataset

[5] F. Chollet et al., Keras: Deep Learning for Python. [Online]. Available: https://keras.io

[6] S. Daware, "Text to Handwriting," GitHub, 2020. [Online]. Available: https://github.com/saurabhdaware/text-to-handwriting

[7] I. J. Goodfellow et al., "Generative Adversarial Nets," in Advances in Neural Information Processing Systems, vol. 27, 2014