



A Baseline Optical Character Recognition Framework for Printed Kashmiri Nastaliq Text Using Deep Learning

Sheikh Amir Fayaz^{1*}, Muzamil Majeed Khaja¹, Abdul Saboor Bhat¹, Danish Mansoor¹, Anu Thapa¹, Majid Zaman²

¹ Directorate of Information Technology & Support System, University of Kashmir, 190023 Srinagar, India

² Department of Computer Sciences, University of Kashmir, 190023 Srinagar, India

* Correspondence: Sheikh Amir Fayaz (skh.amir88@gmail.com)

Received: 01-25-2026

Revised: 03-17-2026

Accepted: 03-26-2026

Citation: S. A. Fayaz, M. M. Khaja, A. S. Bhat, D. Mansoor, A. Thapa, and M. Zaman, "A baseline optical character recognition framework for printed Kashmiri Nastaliq text using deep learning," *Acadlore Trans. Mach. Learn.*, vol. 5, no. 2, pp. 119–137, 2026. <https://doi.org/10.56578/ataiml050203>.



© 2026 by the author(s). Licensee Acadlore Publishing Services Limited, Hong Kong. This article can be downloaded for free, and reused and quoted with a citation of the original published version, under the CC BY 4.0 license.

Abstract: Optical Character Recognition (OCR) plays a crucial role in the digitization and preservation of textual information; however, for low-resource languages such as Kashmiri, reliable OCR solutions remain largely unavailable. Kashmiri, primarily written in the Perso-Arabic (Nastaliq) script, poses significant challenges due to its cursive structure, extensive use of ligatures, complex diacritical marks, and limited availability of annotated datasets. This research aims to address these challenges by developing a functional OCR system specifically tailored for Kashmiri text. The proposed system is built using the open-source Kraken OCR engine and leverages deep learning techniques with transfer learning from a pre-trained Arabic OCR model. A synthetic dataset was generated using Unicode Kashmiri text, enriched with Kashmiri-specific diacritics and exclusive characters, and rendered into images through automated text-to-image pipelines. Extensive preprocessing, augmentation, and iterative fine-tuning were performed to improve recognition accuracy. Model performance was evaluated using standard metrics such as Character Error Rate (CER) and Word Error Rate (WER) on both seen and unseen data. Experimental results demonstrate a substantial improvement over the initial model, with character accuracy increasing from 54.91% to 79.91% and word accuracy improving from 4.65% to 44.19%. The final model shows strong recognition capability for common and Arabic script characters, while Kashmiri-specific inherited diacritics remain a challenging area. In addition, a cross-platform user interface developed using Flutter enables users to upload or capture images and obtain digitized Kashmiri text through a simple and accessible workflow. Rather than proposing a new recognition architecture, this work contributes empirical insights, reproducible methodology, and error characterization for OCR in a previously unsupported low-resource Nastaliq language. This work is positioned as a baseline OCR system for printed Kashmiri Nastaliq text at the line level and does not claim state-of-the-art performance.

Keywords: Optical Character Recognition; Perso-Arabic (Nastaliq) script; Low-resource languages; Deep learning; Transfer learning; Text digitization; Character Error Rate; Word Error Rate

1 Introduction

Background

OCR, or Optical Character Recognition, is not new. It's the technology that lets a computer read text from a page or an image. For English, Chinese, Arabic and many other major languages, this has already become routine. But for Kashmiri, things are different. The language, spoken mostly in the Kashmir Valley [1, 2], has a rich literary tradition but very little digital support.

The Perso-Arabic script, which is widely used for Kashmiri, is complex. It has diacritical marks, variations in shapes, and in some cases different styles of writing the same word. This makes recognition difficult [3]. On top of that, there isn't much annotated data to train on. That's where tools like Kraken OCR come in. Kraken is open-source, customizable, and allows us to train models for low-resource languages [4]. By working with Kraken [5], this research aims to create a Kashmiri OCR system. The idea is simple: start small, get something working, and then improve overtime.

Problem Statement

Right now, there is no reliable OCR system for Kashmiri. If someone wants to digitize a book, a poem, or even an old manuscript, they usually end up typing it manually. That’s slow, tiring, and prone to mistakes.

The problems are clear:

1. No pre-trained OCR models exist for Kashmiri.
2. Script complexity and diacritical marks confuse most generic OCR engines. Very few properly labelled datasets are available.

Because of these issues, digitization is stuck. Scholars, students, or even ordinary readers don’t get easy access to Kashmiri texts in digital form. This gap shows why a dedicated OCR solution is badly needed.

Objectives

The main goals of this study are:

1. Build a working OCR model for Kashmiri using the Kraken engine.
2. Train the model on Kashmiri data to get better recognition accuracy.
3. Test it on both printed and handwritten samples to see how well it performs.
4. Lay the foundation for future work—whether that means handwriting recognition, multilingual support, or NLP integration.

Alongside these, the research also hopes to contribute to the digital presence of Kashmiri and give researchers a tool they can start experimenting with.

Scope

This study focuses exclusively on printed Kashmiri text written in the Perso-Arabic (Nastaliq) script and performs recognition at the line level. The proposed system is intended as a baseline OCR framework for low-resource Kashmiri text rather than a fully optimized or production-ready solution.

Significance of the Study

This work matters for more than one reason. Culturally, Kashmiri has centuries of poetry, history, and thought written in its script. Much of it is at risk of being lost or left behind in dusty libraries. Digitization can preserve it for the future.

Academically, having a Kashmiri OCR opens doors for researchers and linguists. They can study texts faster, analyse patterns, and build new tools on top of the digitized material [6, 7]. From a technology point of view, this research is a start. With it, we can imagine digital Kashmiri libraries, screen readers for accessibility, and even NLP applications like translation or text-to-speech. The system may not solve everything right away, but it begins to close a gap that has existed for too long. And this creates a pathway for future works and contributes to the field significantly. As the era is progressing the writing method is likely to become obsolete endangering the literature and that has not been digitized so far.

So, this study if not gives the optimum working OCR but still lays the foundations of OCR for Kashmiri text and other low resource languages. giving a particular pathway to inculcate the issues and challenges and more importantly how to resolve and approach the field specific issues.

This study does not introduce a novel OCR architecture; instead, it provides a carefully analyzed and reproducible baseline for printed Kashmiri Nastaliq OCR. The main contributions are: (i) the first end-to-end OCR pipeline explicitly tailored for printed Kashmiri Nastaliq text using the Kraken engine, (ii) an empirical evaluation of transfer learning from Arabic to Kashmiri Nastaliq, quantifying its effectiveness in a low-resource setting, (iii) systematic ablation experiments that reveal the relative impact of augmentation and preprocessing strategies, and (iv) detailed character-level and diacritic-specific error analyses that highlight script-specific challenges and guide future research.

2 Literature Review

A comprehensive review of OCR for the Kashmiri language: scripts, challenges, and methodological advancements.

2.1 Introduction: The Linguistic and Orthographic Landscape of Kashmiri

The digitization of cultural heritage and the enhancement of data accessibility through OCR have advanced significantly, yet the development of robust systems for many Indic languages remains a complex challenge [8]. This is particularly true for Kashmiri, a language with a rich linguistic history and a unique multi-script orthography [9]. As one of the 22 scheduled languages of India and a member of the Dardic group of the Indo-Aryan language family, Kashmiri possesses a distinct set of characteristics that complicate the OCR process [9].

The primary obstacle to a monolithic OCR solution for Kashmiri is its use of three fundamentally different writing systems [9]. The official script, recognized by the Jammu and Kashmir government, is the Perso-Arabic script, predominantly utilized by the Muslim community [9]. Hindus, in contrast, often use the Devanagari script for writing the language [9]. The historical context is further enriched by the traditional Sharada script, which was the main literary and inscriptional script for centuries and is still in use today by a small community of Kashmiri

Pandits for liturgical and ceremonial purposes [3, 9]. This orthographic diversity means that a single OCR engine is insufficient. Instead, a tri-modal system is required, with each module tailored to the specific complexities of its corresponding script.

The multi-script nature of Kashmiri is a foundational consideration that influences every aspect of OCR research, from data collection to model design. The prevalence and official status of a script directly impact the availability of digital corpora, which are a prerequisite for training modern machine learning models. The official Perso-Arabic script, used in contemporary texts like newspapers and magazines, has a higher volume of available data for corpus creation [9]. Conversely, the Sharada script's limited use has resulted in a critical lack of standardized, publicly available datasets, which serves as a major bottleneck for any meaningful OCR development [1, 2, 9]. This disparity in resources means that research efforts and advancements are not uniform across all three scripts.

2.2 The Inherent Challenges of Optical Character Recognition for Kashmiri Scripts

The difficulty of OCR for the Kashmiri language is not a general problem but a set of distinct, script-specific challenges. The structural properties of each writing system necessitate unique approaches and present different obstacles to accurate recognition [10].

The Perso-Arabic script for Kashmiri is principally written in the Nastaliq style, which is known for its highly cursive nature and sloping baseline. Unlike most Latin-based scripts, Nastaliq is written from right to left, and its characters lack case distinction [11]. The primary recognition challenge stems from its use of ligatures, which are calligraphic units formed by cursively joining two or more characters [11]. A word in Nastaliq is not a continuous sequence of individual characters but rather a composition of these ligatures and isolated characters, which are often referred to as sub-words [11].

The context-sensitive nature of the script further complicates recognition. A character's shape changes radically depending on its position within a word (initial, medial, final, or isolated) and the characters to which it joins [11]. These multiple forms, or glyphs, for a single character introduce a high degree of variability. OCR systems must correctly identify these contextual shapes to produce the correct character code. Another significant challenge is the correct association of diacritics and vowel marks, which are secondary components, with their primary ligature bodies [9]. Kashmiri has 16 vowel sounds, far more than in Arabic or Persian, and all diacritics are always visible, though their form may change based on position. Finally, the aesthetic quality of Nastaliq, characterized by the intentional reduction of space between character pairs (kerning) to improve visual appeal, can cause slight character overlapping [11]. When scanned, this lack of white space can lead OCR systems to misread two distinct characters as a single, continuous shape, resulting in recognition errors [11]. These characteristics make traditional character-by-character segmentation approaches highly prone to error and have led researchers to explore alternative, segmentation-free methods [12].

2.3 State-of-the-Art Optical Character Recognition Methodologies and Their Application to Kashmiri

The inherent complexities of Kashmiri's scripts have driven research to explore a range of OCR methodologies, from traditional statistical models to cutting-edge deep learning and large language model approaches. The most relevant developments in the field have often come from OCR research for structurally similar languages, most notably Urdu, which shares the Nastaliq script [13, 14].

2.3.1 Foundational Optical Character Recognition paradigms

OCR for complex scripts can be broadly categorized into two main paradigms: segmentation-based and segmentation-free. The segmentation-based approach, which is common for Latin scripts, involves breaking down the text region into lines, words, and then individual characters. For cursive scripts like Nastaliq, this method is particularly challenging and error-prone due to the connected nature of the writing and the presence of ligatures [12]. The segmentation-free approach, which treats ligatures as the basic units of recognition, has proven more effective for Nastaliq [12, 15]. This method simplifies the initial stages of the OCR pipeline by bypassing the difficult task of character-level segmentation. However, this comes at a cost: it significantly increases the number of classes that the OCR system must learn to recognize, as there are thousands of unique ligatures.

2.3.2 Traditional and modern machine learning approaches

Early research on Nastaliq OCR utilized Hidden Markov Models (HMMs) for segmentation-free recognition of ligatures [15]. This approach was a significant step forward from character-based methods, as it correctly identified the ligature as the primary unit of recognition [15]. With the advent of deep learning, a new generation of models has been applied to the problem. Recurrent Neural Networks (RNNs) and their variants, such as Long Short-Term Memory (LSTM) networks, have shown superior performance for cursive and handwritten script recognition. One study on Urdu Nastaliq reported that an LSTM-based system achieved an average character accuracy of 97.93% and a ligature accuracy of 96.19% [16]. Convolutional Neural Networks (CNNs) have also been used, often in conjunction

with LSTMs in a hybrid architecture. In Devanagari OCR, a CNN-based technique has been proposed with the objective of achieving 96% accuracy [17].

Classical machine learning classifiers, such as Support Vector Machines (SVMs), have also been employed, typically in the classification stage of an OCR pipeline after feature extraction. For instance, an SVM classifier was used with a Histogram of Oriented Gradients (HOG) feature extraction method for Devanagari recognition, yielding a high accuracy of 95.9% [16]. A ligature-based segmentation system for Urdu Nastaliq using an SVM classifier achieved an overall accuracy of 90.29% [12].

2.3.3 The emergence of Large Language Models (LLMs)

The most recent and promising advancements have involved the fine-tuning of LLMs for OCR. A new paper introduces a pipeline that uses fine-tuned LLMs for text recognition from images of Urdu newspapers [18]. This approach has shown remarkable results, with a model like Gemini-2.5-Pro achieving a Word Error Rate (WER) of 0.133, a significant improvement over traditional method [2]. This suggests that LLMs, with their ability to understand complex patterns and contexts, may be well-suited to overcome some of the long-standing challenges of Nastaliq script recognition, particularly in noisy, real-world conditions.

2.4 Data Resources and Performance Metrics

The success of any OCR system is directly contingent on the availability of high-quality training and testing data. An understanding of the existing corpora and the metrics used to evaluate performance is essential for a comprehensive review of the field.

2.4.1 Existing corpora and benchmarks

The Central Institute of Indian Languages (CIIL) has developed a “Gold Standard Kashmiri Raw Text Corpus”, a significant resource containing 466,054 words and over 2.6 million characters from 108 titles [19]. The corpus is drawn from diverse sources like books, newspapers, and magazines, covering domains such as Aesthetics and Social Sciences [19]. This data is in Unicode and XML format, which is highly valuable for linguistic analysis and language modeling [19]. However, it is a critical distinction that this is a raw text corpus, meaning it consists of digitized text files without the corresponding document images [19]. For training an OCR system, which requires image-to-text mapping, a corpus of image-text pairs is necessary. Therefore, while the LDC-IL corpus is a monumental step for Kashmiri NLP, it is not a direct resource for OCR training [19].

The lack of a dedicated image-based corpus for Kashmiri has led researchers to rely on resources from related languages. The Urdu-Nastaliq Handwritten Dataset (UNHD) is a publicly available dataset for handwritten Urdu, providing a comprehensive collection of ligatures written by 500 writers [20]. For printed text, the Urdu Newspaper Benchmark (UNB) provides a realistic, manually annotated testbed of noisy, multi-column newspaper scans [18]. These datasets serve as crucial proxies, enabling the transfer of models and methodologies from Urdu to Kashmiri Nastaliq.

2.4.2 Evaluating Optical Character Recognition performance

The performance of OCR systems is typically measured using standard metrics, most commonly WER and Character Error Rate (CER). These metrics quantify the number of insertions, deletions, and substitutions required to transform the OCR output into the ground-truth text, with a lower error rate indicating higher accuracy. The reported accuracies in the literature show a wide range. For Nastaliq, a ligature-based system achieved an overall accuracy of 90.29% [12], while an LSTM-based model reported high character-level accuracy of 97.93% [17]. More recent work using LLMs on the UNB, a complex, real-world dataset, achieved a WER of 0.133, which corresponds to an 86.7%-word accuracy [18]. For Devanagari, a literature survey noted an average accuracy of 94.2% for different methods, with some approaches aiming for a 96% accuracy rate [16].

The discrepancy between the high accuracy rates of some studies and the lower rates on real-world data can be attributed to the nature of the datasets. Models trained and tested on highly curated or synthetic data, such as a dataset of isolated ligatures, tend to report very high accuracy. In contrast, systems evaluated on noisy, unconstrained, and complex documents, like a scanned newspaper page, will inevitably have lower performance metrics but provide a more realistic measure of a system’s true capabilities.

2.5 Synthesis of Findings and Future Research Directions

The review of the literature on OCR for Kashmiri reveals a field with significant progress in some areas but substantial gaps in others. The primary challenge is not a single technical hurdle but the multi-script nature of the language, which requires a segmented approach to research and development [9]. For the official Perso-Arabic (Nastaliq) script, the field has matured from traditional segmentation-based methods to ligature-based systems using HMMs and, more recently, to advanced deep learning models like LSTMs and LLMs [15]. The advancements in Urdu OCR serve as a reliable indicator for the potential of these methodologies to tackle the unique complexities of

Nastaliq, which include its cursive nature, sloping baseline, and context-sensitive glyphs [11]. The development of specialized Urdu datasets, such as the UNHD and UNB, demonstrates that the creation of high-quality corpora is the first and most crucial step in training robust models [18, 20].

Future work in this area should be guided by a few key priorities. The most immediate is the creation of comprehensive, publicly available image-text corpora for all three scripts, especially for Sharada. This is a foundational prerequisite for applying the sophisticated machine learning models that have proven successful with other languages. Furthermore, the exploration of LLMs for their potential to handle the contextual and layout complexities of Nastaliq, as demonstrated by the recent Urdu OCR research, warrants further investigation [18]. Finally, any robust system for Kashmiri must incorporate a mechanism for automatically identifying the script within a document, allowing it to route the image to the appropriate, specialized OCR engine.

Thus, OCR for the Kashmiri language is a multi-faceted problem defined by the distinct challenges of its three primary scripts [9]. The official Nastaliq script presents difficulties related to its cursive and ligature-heavy nature, as demonstrated by advancements in related fields like Urdu OCR, modern deep learning models, including CNNs, LSTMs, and even LLMs, offer promising pathways to high-accuracy recognition [17, 18]. However, these technological solutions are dependent on foundational data resources that are currently scarce or non-existent for key orthographies. The digitization of Kashmiri’s linguistic heritage through OCR is not just a technological objective but a critical act of cultural preservation that requires the concerted effort of academic institutions, government bodies, and the community to address the identified data and methodological gaps.

3 Methodology

The methodology adopted for developing the Kashmiri OCR system is presented in detail. The process encompasses four main stages: dataset preparation, preprocessing, model training, and testing and evaluation. Each stage has been designed to ensure accurate recognition of Kashmiri text, including the handling of complex diacritics inherent in the script.

3.1 Dataset Preparation

The first step involved in collecting source text in Unicode Kashmiri from multiple websites including Wikipedia as the text should be encoded in UTF-8, as Kashmiri being a low resource language it gets challenging to get the data. The data we collected so far was mostly from “kashmirilanguage.com” as there were multiple formats for the data including pdf, the data we acquired was pure text. Each textual data batch is roughly 100 lines of text rich in diacritics including Arabic diacritics such as (Tables 1-3). To prevent data leakage, dataset splitting was performed at the level of unique text lines prior to any augmentation. The dataset was first divided into training, validation, and test sets, and augmentation was applied independently within each split. This ensures that no augmented variants of the same textual content appear across different subsets.

To improve diacritic recognition, additional training samples rich in Kashmiri inherited diacritics were incorporated during dataset construction, and their frequency was better balanced across the training data. This was intended to reduce bias toward frequently occurring base characters and expose the model more consistently to Kashmiri-specific vowel and accent marks.

3.1.1 Arabic diacritics (Harakat)

These are marks added to letters to indicate short vowels.

Table 1. Arabic diacritics

Diacritic	Name (Arabic)
َ	Fathah (فَتْحَة)
ِ	Kasrah (كَسْرَة)
ُ	Dammah (ضَمَّة)
َ	Tanwīn Fath (تَنْوِين فَتْح)
ِ	Tanwīn Kasr (تَنْوِين كَسْر)
ُ	Tanwīn Dam (تَنْوِين ضَم)
◌◌◌	Sukūn (سُكُون)
◌◌◌◌	Shaddah (شَدَّة)
◌◌◌◌◌	Alif Waslah/Maddah (مَدَّة)

3.1.2 Kashmiri diacritics

Kashmiri uses extended Arabic script (Perso-Arabic based) and adds some diacritics to represent vowels and consonants that Arabic doesn't have.

Table 2. Kashmiri diacritics

Diacritic	Description
-	Sukūn—absence of vowel
و	Waw with small ring—represents vowel /o/
و	Represents vowel /ö/ or /ø/

3.1.3 Kashmiri exclusive letters

Kashmiri not just uses Arabic characters but includes Urdu based characters and some Kashmiri exclusive letters as well such as:

Table 3. Kashmiri exclusive letters

Letter	Name/Description
ژ	Zheh
ڑ	Ddieh
ٹ	Tteh
چھ	Chheh
ڈ	Ddal
ں	Nūn Ghunna

Table 4 summarizes the dataset composition used in this study, including the number of unique Kashmiri text lines, the number of original rendered images, the number of augmented images, and the exact training, validation, and test split. Dataset splitting was performed prior to augmentation to avoid data leakage.

Table 4. Dataset composition and split details

Dataset Split	Unique Text Lines	Original Images	Augmented Images	Total Images
Training	750	750	3,000	3,750
Validation	100	100	400	500
Test (Synthetic)	150	150	0	150
Total	1,000	1,000	3,400	4,400

3.2 Preprocessing

Prior to model training the Kraken(engine) expects a ground-truth file in formats (.gt.txt / XML/ ALTO) and corresponding image file in formats (TIFF/ PNG/ JPEG) [12, 13]. For this the process that the text went through is as follows.

3.2.1 Data cleaning

The textual data collected contained unnecessary characters such as numerals, special characters, English alphabet letters and empty lines. The data was cleaned using simple PowerShell script:

```
“(Get-Content file.txt) | ForEach-Object { $_ -replace '[A-Za-z0-9[:punct:]]', '' } | Set-Content cleaned.txt”
```

3.2.2 Ground truth file generation

The cleaned data was gathered in to a single file then the file was sliced into individual single line files and saved as .gt.txt files serially, this was done also using a simple PowerShell script:

```
$lines = Get-Content “cleaned.txt”
for ($i = 0; $i -lt $lines.Count; $i++) {
$fileName = “{0:D2}.gt.txt” -f ($i + 1) 01.gt.txt, 02.gt.txt, etc.
Set-Content -Path $fileName-Value $lines[$i]
}
```

3.2.3 Image generation

For each ground-truth file we must have an image corresponding to it, but the collected data had no ready to use images. So, the images were synthetically generated using text2image tool provided by the Tesseract OCR engine (Google, 2024) [6]. The tool was fed with .gt.txt files using the following Bash script (Table 5):

```
for file in .gt.txt; do
base="${file%.gt.txt}"
text2image -text "$file" -outputbase "$base" -font "Gulmarg Nastaleeq"
-fonts_dir ./fonts -ptsize 42 -max_pages 1 -leading 32 -char_spacing 0
-xsize 3600 -ysize 480 -resolution 300 \
Done
```

Table 5. Text2 image command params description

Parameter	Description
-text "Sfile"	Specifies the input text file (UTF-8) to convert into an image. Each .gt.txt file contains a single line of Kashmiri text.
-outputbase "Sbase"	Sets the base name for the output image file. The resulting TIFF/PNG file will have this base name (same as the input text).
-font "Gulmarg Nastaleeq"	Selects the font for rendering the text. This font closely resembles traditional Kashmiri Nastaleeq style.
-fonts dir /fonts	Directory path containing the font files. Ensures text2image can locate the Gulmarg Nastaleeq font.
-ptsize 42	Point size of the font (larger point size improves readability and preserves diacritics in synthetic images).
-max pages 1	Limits the output image to one page per input file (i.e., a single line of text per image).
-leading 32	Sets line spacing in pixels. For single-line images, it controls vertical positioning of the text.
-char spacing 0	Sets additional spacing between characters. 0 means default spacing; can be increased for clarity.
-resolution 300	DPI (dots per inch) of the output image. Higher DPI gives sharper text and improves OCR accuracy.

The image produced by the tool using the above parameters for the text “بہ کیا ونے ژے مے کیتھ وچہ عجیب خاب” produces the following image (Figure 1):

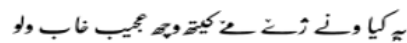


Figure 1. Rendered image using script shown in Table 5

After rendering the image, the parameters are tweaked like -ptsize by changing the values to mimic different font sizes, -resolution changing it to render different resolution images so that the model gets used to different types of inputs. As the engine relies on GPU and is time consuming for the rendered image dimensions, the unnecessary blank portion of the image is trimmed to save time and increase throughput. For that we used the utility tool imagemagick by ImageMagick Studio LLC [7]. For the rendered file the trimming was done using imagemagick's [8] CLI:

```
Get-ChildItem .tif | ForEach-Object { magick $_.FullName -trim -bordercolor White -border 25 "$($_.FullName)" }
```

The resultant image is as Figure 2.

All the Images rendered are in binary form '0' representing white pixel and '1' representing black pixel. Each batch containing around 100 files is fed to the Kraken engine to train a model then the same batch of files are augmented using the same tool to increase noise and other properties like rotation, skewing etc. (Figure 3).

یہ کیا ونے ژے مے کیتھ وچھ عجیب غاب ولو

Figure 2. Trimmed image using Magick tool

```
$magick file.tif -rotate -5 file_rotated.tif
```

یہ کیا ونے ژے مے کیتھ وچھ عجیب غاب ولو

Figure 3. Rotated image using Magick tool -5 deg

Similarly, each batch goes through different augmentations like noise, skew etc.

```
$ magick file.tif -attenuate 0.1 +noise Gaussian file_noise.tif
```

```
$ magick file.tif -rotate 2 -attenuate 0.1 +noise Gaussian file_aug.tif
```

3.3 Model Training

The model training is core of the research as everything revolves around the model. The training took the longest time among all the phases of development. As the data for model training was prepared it was time to setup the training tools.

3.3.1 Environment configuration

The Kraken engine relies heavily on GPU as the system used for training was equipped with NVIDIA GPU to accelerate deep learning computations. Alongside Kraken is based on Python 3.9 with the following library: PyTorch for neural network training. Kraken 3.5.2 was installed through Python Package Index using Windows Subsystem for Linux (WSL2).

Table 6. Ketos command description

Parameter	Description	Role in Training
-v	Verbose mode	Provides detailed log output during training, including loss updates and progress indicators.
train	Sub-command	Instructs Kraken to enter model training mode.
-resize union	Line dimensions	Adjusts each image's dimensions to fit with rest by adding or trimming white border.
-device cuda:0	Device specification	Allocates training to the first GPU (index 0). If no GPU is available, it can be set to CPU.
-workers 4	Number of data loading workers	Uses 4 CPU threads for loading and preprocessing data in parallel, speeding up training.
-epochs 10	Number of training epochs	Runs 10 full passes over the dataset. Controls the duration of training and convergence depth.
-batch size 16	Mini-batch size	Processes 16-line images per training step. Larger batches improve stability but require more GPU memory.
-force binarization	Image preprocessing option	Converts grayscale or color images into binary (black and white). Useful for noisy scans to improve recognition.
segment	Training mode	Uses segmentation-based training, meaning the dataset includes segmented line-level inputs.
-o kashmiri	Output model name	Saves the trained model as kashmiri.mlmodel.
-load arabic best.mlmodel	Pretrained model initialization	Loads an existing Arabic model as a starting point (transfer learning). Helps adapt to Kashmiri with less data.
.tif	Training data input	Uses all .tif files in the current directory as training images. Each must have a matching .gt.txt file.

3.3.2 Training procedure

Kraken primarily provides two training interfaces.

- a. `kraken -train`: the interface takes input data only in XML format.
- b. `ketos -v train`: this interface takes `.gt.txt` for training [5].

`kraken -train` being deprecated and dependent on XML data the training was carried out using `ketos` interface. Although the Kraken gives freedom of training and creating a model from scratch, the training was firstly based on pretrained model of Arabic language `arabic_best.mlmodel`. Using a pretrained model gave the head start for model creation and understanding of ligatures was comparatively easy for the model. Ketos CLI command structure and log:

`$ketos -v train -resize union -device cuda:0 -workers 4 -epochs 10 -batch_size 16 -force_binarization segment -o kashmiri -load arabic_best.mlmodel.tif` (Table 6)

Grapheme detection LOG:

```

08/21/25 18:04:01 INFO Loading existing model from ../arabic_best.mlmodel train.py:237
08/21/25 18:04:03 WARNING Forced binarization enabled in 'path' mode. Will be ignored. train.py:298
INFO Got 91 line strip images for training data train.py:303
INFO No explicit validation data provided. Splitting off 10 (of 91) samples to train.py:307
INFO validation set. (Will disable alphabet mismatch detection.)
INFO Training set 81 lines, validation set 10 lines, alphabet 53 symbols train.py:410
INFO grapheme count train.py:421
INFO SPACE 463 train.py:426
INFO ا 209 train.py:426
INFO آ 177 train.py:426
INFO ع 165 train.py:426
INFO ح 147 train.py:426
INFO ه 126 train.py:426
INFO ج 126 train.py:426
INFO ق 106 train.py:426
INFO ف 85 train.py:426
INFO ARABIC KASRA 84 train.py:426
INFO ك 78 train.py:426
INFO گ 72 train.py:426
INFO غ 70 train.py:426
INFO ARABIC DAMMA 65 train.py:426
INFO ي 64 train.py:426
INFO و 51 train.py:426
INFO ARABIC HAMZA BELOW 47 train.py:426
INFO ج 45 train.py:426
INFO ح 41 train.py:426
INFO خ 37 train.py:426
INFO د 35 train.py:426
INFO ذ 35 train.py:426
INFO ر 30 train.py:426
INFO ARABIC HAMZA ABOVE 29 train.py:426
INFO و 29 train.py:426
INFO ARABIC VOWEL SIGN SMALL V ABOVE 29 train.py:426
INFO و 27 train.py:426
INFO و 26 train.py:426
INFO و 24 train.py:426
INFO و 23 train.py:426
INFO ARABIC FATHA 22 train.py:426
INFO و 19 train.py:426
INFO و 16 train.py:426
INFO و 14 train.py:426
INFO و 12 train.py:426
INFO ARABIC VOWEL SIGN INVERTED SMALL V ABOVE 12 train.py:426
INFO و 10 train.py:426
INFO و 9 train.py:426
INFO و 7 train.py:426
INFO ARABIC INVERTED DAMMA 7 train.py:426
INFO و 6 train.py:426
INFO و 5 train.py:426
INFO و 5 train.py:426
INFO و 3 train.py:426
INFO و 2 train.py:426
INFO و 2 train.py:426
INFO و 2 train.py:426
INFO ARABIC SUBSCRIPT ALEF 1 train.py:426
INFO و 1 train.py:426
INFO و 1 train.py:426
INFO ARABIC SHADDA 1 train.py:426
INFO و 1 train.py:426
INFO Encoding training set train.py:430

```

Figure 4. Kraken input data detection

In the above Figure 4, the engine is fed with 91 data lines, out of which 10% is implicitly set for validation to measure CER and WER, and to avoid overfitting and underfitting. The Grapheme and Count show each character's frequency throughout the dataset. The learning curve of the model depends highly on balanced character frequency. In the above picture, it is clear that some characters appear more frequently, giving the edge of learning for only certain characters to be recognized by the model. This creates an imbalance to the model, so a model based on balanced saturation of character appearance learns better.

Learning LOG is shown in Figure 5.

The log of the initial model generation and performance stats based on `arabic_best.mlmodel` is displayed in Figures 5–6. It is clear that the model increases in accuracy and decreases in error with each iteration of model generation up to a fixed number of epochs, in this case 12 epochs. Based on the results of the validation run, the best model (`kashmiri.mlmodel`) is chosen with the highest accuracy after the entire run.

The output of the first run, `kashmiri.mlmodel`, serves as the base for the second run. In this manner, the model learns all of the modifications that new data brings. And with every batch of data that is fed into the engine, the fine-tuning continues.

Training log of best model (Figure 7).

Comparison of initial model with latest finetuned model based on accuracy and errors over epochs:

The aforementioned graphs in Figures 8–9 make it evident how the model learns and gets better over time as the error rate drops. Despite how abrupt the change may appear; it ends around 15 models later. Therefore, compared to individual model generation, the changes were minimal.

```

Non-trainable params: 0
Total params: 411 W
Total estimated model params size (MB): 16
[08/21/25 10:04:07] INFO Upgrading "is_model" from b'1' to b'L'
[08/21/25 10:07:14] INFO validation run: total chars 316.0 errors 136.0 accuracy 0.5696202516555786
train.py:528
INFO Saving to kashmiri_0.mlmodel
train.py:194
[08/21/25 10:07:15] INFO Setting model one_channel_mode to L.
train.py:179
[08/21/25 10:09:49] INFO validation run: total chars 316.0 errors 122.0 accuracy 0.6139240264892578
train.py:528
INFO Saving to kashmiri_1.mlmodel
train.py:194
[08/21/25 10:12:30] INFO validation run: total chars 316.0 errors 130.0 accuracy 0.5886076092720032
train.py:528
INFO Saving to kashmiri_2.mlmodel
train.py:194
[08/21/25 10:15:03] INFO validation run: total chars 316.0 errors 123.0 accuracy 0.6107594966888428
train.py:528
INFO Saving to kashmiri_3.mlmodel
train.py:194
[08/21/25 10:17:41] INFO validation run: total chars 316.0 errors 120.0 accuracy 0.6202532052993774
train.py:528
INFO Saving to kashmiri_4.mlmodel
train.py:194
[08/21/25 10:20:15] INFO validation run: total chars 316.0 errors 119.0 accuracy 0.6234177350997925
train.py:528
INFO Saving to kashmiri_5.mlmodel
train.py:194
[08/21/25 10:23:01] INFO validation run: total chars 316.0 errors 119.0 accuracy 0.6234177350997925
train.py:528
INFO Saving to kashmiri_6.mlmodel
train.py:194
[08/21/25 10:25:35] INFO validation run: total chars 316.0 errors 115.0 accuracy 0.6360759735107422
train.py:528
INFO Saving to kashmiri_7.mlmodel
train.py:194
[08/21/25 10:28:13] INFO validation run: total chars 316.0 errors 111.0 accuracy 0.6487342119216919
train.py:528
INFO Saving to kashmiri_8.mlmodel
train.py:194
[08/21/25 10:30:51] INFO validation run: total chars 316.0 errors 107.0 accuracy 0.6613923907279968
train.py:528
INFO Saving to kashmiri_9.mlmodel
train.py:194
[08/21/25 10:33:36] INFO validation run: total chars 316.0 errors 106.0 accuracy 0.6645569801330566
train.py:528
INFO Saving to kashmiri_10.mlmodel
train.py:194
[08/21/25 10:36:23] INFO validation run: total chars 316.0 errors 99.0 accuracy 0.6867088675498962
train.py:528
INFO Saving to kashmiri_11.mlmodel
train.py:194
[08/21/25 10:38:57] INFO validation run: total chars 316.0 errors 103.0 accuracy 0.6740586291389465
train.py:528
INFO Saving to kashmiri_12.mlmodel

```

Figure 5. Kraken model generation log

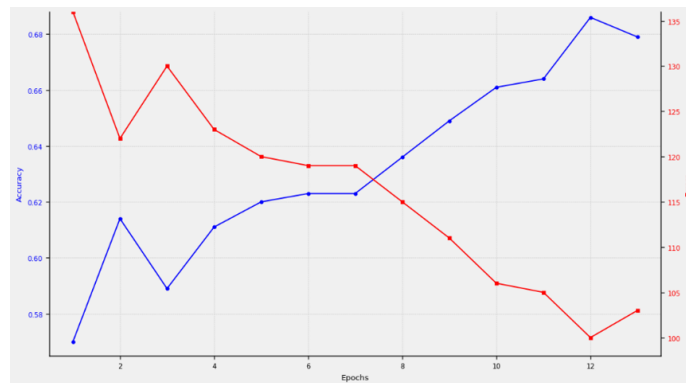


Figure 6. Optical Character Recognition (OCR) accuracy and error over epochs

```

[08/22/25 18:04:55] INFO validation run: total chars 23.0 errors 2.0 accuracy 0.9130434989929199
INFO Saving to kashmiri_0.mlmodel
INFO Setting model one_channel_mode to L.
[08/22/25 18:04:57] INFO validation run: total chars 23.0 errors 3.0 accuracy 0.8695652484893799
INFO Saving to kashmiri_1.mlmodel
[08/22/25 18:04:58] INFO validation run: total chars 23.0 errors 3.0 accuracy 0.8695652484893799
INFO Saving to kashmiri_2.mlmodel
[08/22/25 18:05:00] INFO validation run: total chars 23.0 errors 2.0 accuracy 0.9130434989929199
INFO Saving to kashmiri_3.mlmodel
[08/22/25 18:05:01] INFO validation run: total chars 23.0 errors 3.0 accuracy 0.8695652484893799
INFO Saving to kashmiri_4.mlmodel
[08/22/25 18:05:03] INFO validation run: total chars 23.0 errors 3.0 accuracy 0.8695652484893799
INFO Saving to kashmiri_5.mlmodel
[08/22/25 18:05:05] INFO validation run: total chars 23.0 errors 3.0 accuracy 0.8695652484893799
INFO Saving to kashmiri_6.mlmodel
[08/22/25 18:05:06] INFO validation run: total chars 23.0 errors 3.0 accuracy 0.8695652484893799
INFO Saving to kashmiri_7.mlmodel
[08/22/25 18:05:08] INFO validation run: total chars 23.0 errors 3.0 accuracy 0.8695652484893799
INFO Saving to kashmiri_8.mlmodel
[08/22/25 18:05:09] INFO validation run: total chars 23.0 errors 3.0 accuracy 0.8695652484893799
INFO Saving to kashmiri_9.mlmodel
[08/22/25 18:05:11] INFO validation run: total chars 23.0 errors 3.0 accuracy 0.8695652484893799
INFO Saving to kashmiri_10.mlmodel
INFO Moving best model kashmiri_0.mlmodel (0.9130434989929199) to kashmiri_best.mlmodel

```

Figure 7. Latest model generation log

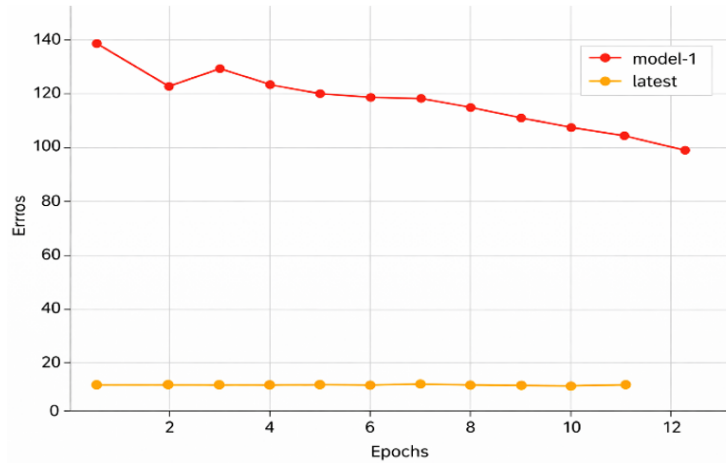


Figure 8. First and latest Optical Character Recognition (OCR) model’s error comparison

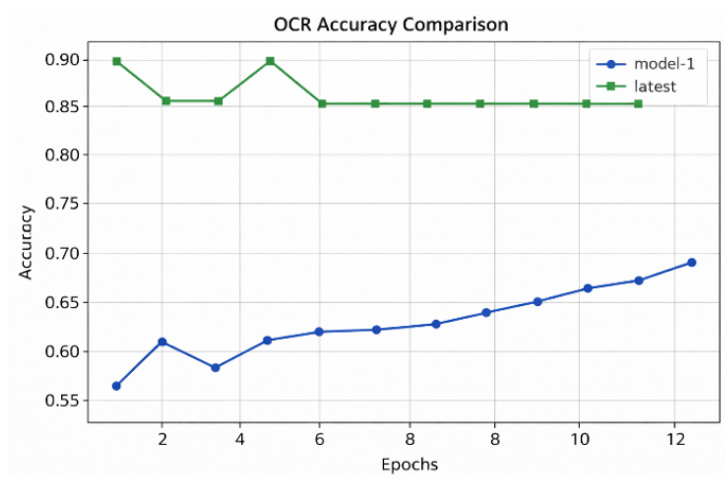


Figure 9. First and latest Optical Character Recognition (OCR) model accuracy comparison

4 Experimental Evaluation and Results

Testing is one of the crucial stages in development to check whether the product is performing the way it was intended to. And the evaluation is necessary to see how accurately and efficiently the product is working. By measuring the WER and CER to validate its accuracy.

After successful model generation each of the models were put to the test to evaluate model accuracy. For that about 20% of the data was used for testing.

4.1 Testing (Seen and Unseen Data)

The first approach of testing was done manually by feeding an image containing Kashmiri text to the model to if the model was working (Figure 10):

Model-1: \$ kraken -i ../file10.tif ../out.txt binarize segment ocr -m ../first.mlmodel

Image input (seen data):

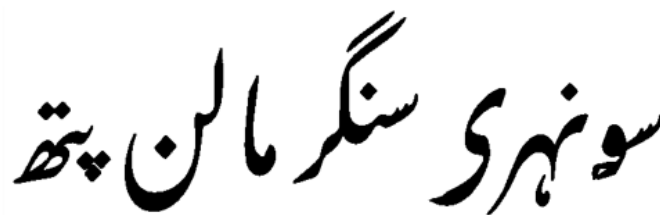


Figure 10. Input data to first model

Output text: سونہری سنگر مالن پتھ

It is clear after observing the output of the given image the model struggles to distinguish the Kashmiri specific diacritics and characters. As the training progresses the model slowly starts to identify the ligatures pertaining to the Kashmiri language.

Model-Latest: \$ kraken -i ../file10.tif ../out1.txt binarize segment ocr -m kashmiri.mlmodel

Input image (seen data) (Figure 11):



Figure 11. Input image to latest model

Output text: سونہری سنگر مالن پتھ

The most recent model recognizes every character in the image as seen above, including those unique to Kashmir, but it still has trouble with a few diacritical marks. Additionally, it occasionally omits the white spaces because in Perso-Arabic script, character boundaries overlap, making it difficult for the model to distinguish between them. It can be eliminated by using more data to teach model how the ligature's characters appear consequently but due to the scarcity of the data, the issue persists remain unattended.

The process is similar as it was for seen data, only the image input fed to the model was randomly selected from the web to test the model.

Model-1: \$ kraken -i ../unseen.png ../out.txt binarize segment ocr -m ../first.mlmodel

Input image (unseen data) (Figure 12):

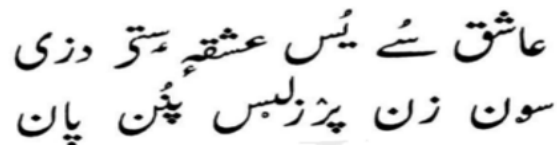


Figure 12. Unseen image input to first model

Output text:

عاست س یس عشقبہ متری دزی
لیم سوان زان پز زلیس پن پان

Model-Latest: \$ kraken -i ../file10.tif ../out1.txt binarize segment ocr -m kashmiri.mlmodel

Input image (Figure 13):

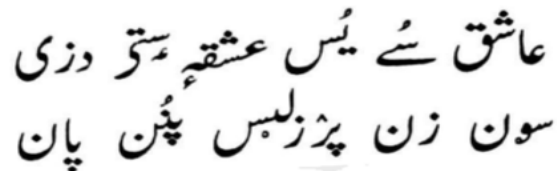


Figure 13. Unseen image input to latest model

Output text:

عاشق سے یس عشقہ متری دزی
سون زن پز زلیس پن پان

4.2 Real World Test Data

To evaluate real-world performance, a separate test set was constructed using scanned pages and mobile phone photographs (Figure 14) of printed Kashmiri books and newspapers. Images were captured under varying lighting conditions, resolutions, and background noise to reflect practical usage scenarios.

Ground-truth transcriptions were manually prepared by native Kashmiri readers. This dataset was used exclusively for testing and was not included in training or validation.

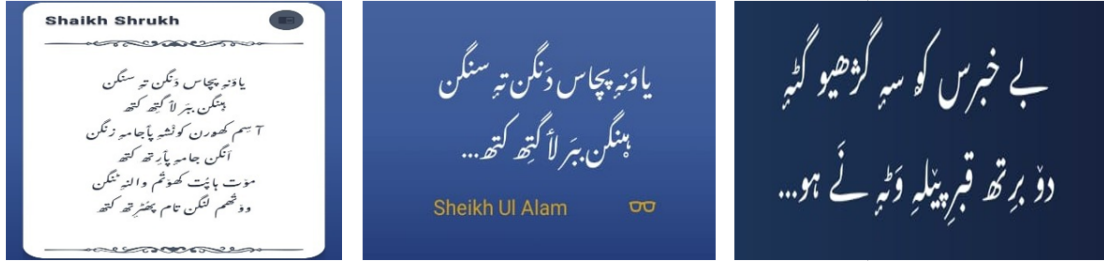


Figure 14. Data set used for testing purposes

4.3 Evaluation

For evaluation the kraken is equipped with ketos-test [5] module to calculate the CER and WER of the model. CER and WER were used as the primary evaluation metrics to assess OCR performance. Prior to evaluation, Unicode normalization was applied to both ground-truth text and OCR output to ensure consistent character representation.

Diacritics were preserved during CER computation in order to accurately measure errors related to Kashmiri-specific vowel and accent marks. For WER calculation, whitespace was explicitly considered to reflect word-boundary and spacing errors commonly observed in Perso-Arabic (Nastaliq) script OCR output (Table 7 and Table 8).

Model-1: \$ ketos test -m first.mlmodel.tif

Table 7. Model 1 performance metrics

Metric	Value
Total Characters	428
Total Errors	193
Character Accuracy	54.91%
Word Accuracy	4.65%

Table 8. Model 1 script-wise Optical Character Recognition (OCR) performance

Script Type	Count	Errors	Accuracy
Common	71	17	76.06%
Arabic	335	150	55.22%
Inherited	22	18	18.18%

4.3.1 Analysis by script type

The performance varied significantly depending on the type of characters (script) being recognized. The script is combination of (Tables 9–11):

- Common: It comprises of Latin, punctuations, special symbols etc.
- Arabic: All Arabic characters fall under this category including Arabic subscript.
- Inherited: Diacritics and vowels that are unionized from Kashmiri language alphabet.

4.3.2 Analysis by script type

Performance improved across the board, but the degree of improvement varies by script (Table 12).

The overall performance measure clearly shows that the model has learned common and Arabic scripts better, compared to inherited script. As mentioned before in Section 3.1 the white-space identification issue increases the WER, still the model has improved significantly as 10 times the first model.

Model evaluation comparison:

Table 9. Analysis of the performance of Model 1

Script Type	Total Count	Errors (Missed)	Accuracy	Analysis
Common	71	17	76.06%	Relatively good performance. The engine is most comfortable with these characters.
Arabic	335	150	55.22%	Poor performance. This is the main script of the text and the primary source of errors, dragging the overall accuracy down.
Inherited	22	18	18.18%	Extremely poor performance. The OCR struggled immensely with fine details like accents.

Table 10. Latest model performance metrics

Metric	Value
Total Characters	428
Total Errors	86
Character Acc	79.91%
Word Accuracy	44.19%

Table 11. Latest model script-wise Optical Character Recognition (OCR) performance

Script	Count	Error	Accuracy
Common	71	5	92.96%
Arabic	335	58	82.69%
Inherited	22	12	45.45%

Table 12. Analysis of the performance of latest model

Script Type	Total Count	Errors (Missed)	Accuracy	Analysis
Common	71	5	92.96%	Excellent performance. Near-perfect recognition of common characters.
Arabic	335	58	82.69%	Major improvement (+27.47%). The core text is now being recognized reliably. This is the key to the overall gain.
Inherited	22	12	45.45%	Significant improvement (+27.27%). While still the weakest area.

From Figure 15, the side-by-side comparison clearly shows how much the latest model performs better in every aspect. Given the model was trained on lesser data it expects to gain accuracy.

Given graph in Figure 16 above shows the models improvement while reducing the error rate in given three fields' Insertions, deletions and substitutions.

Analysis of Figure 15 Given parametric Values

I. Insertions (34 reduced)

Analysis: A reduction of 34 insertions means the model improved by avoiding unnecessary extra symbols. This suggests better segmentation and recognition of character boundaries.

II. Deletions (-3 change)

Analysis: Here, you see -3 (negative reduction), meaning the system slightly worsened in this category, missing a few more characters. This could be due to difficulties with diacritics or faint strokes in handwritten text.

III. Substitutions (76 reduced)

Analysis: A large reduction of 76 substitutions is very positive. It means the model has become much better at distinguishing similar-looking characters (e.g., Arabic/Kashmiri letters like ب vs پ, or diacritics). This shows that training/finetuning had a strong impact here.

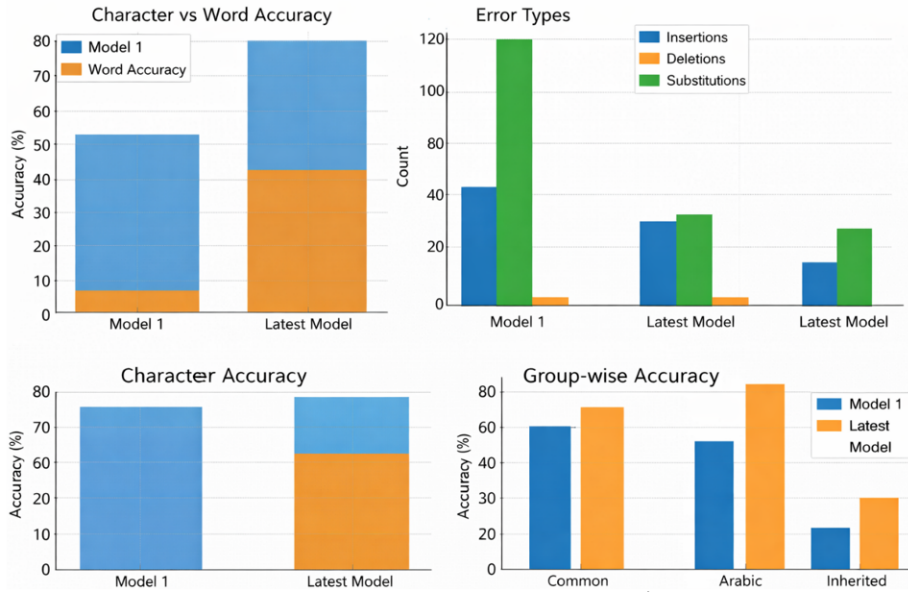


Figure 15. Optical Character Recognition (OCR) latest and first model evaluation comparison

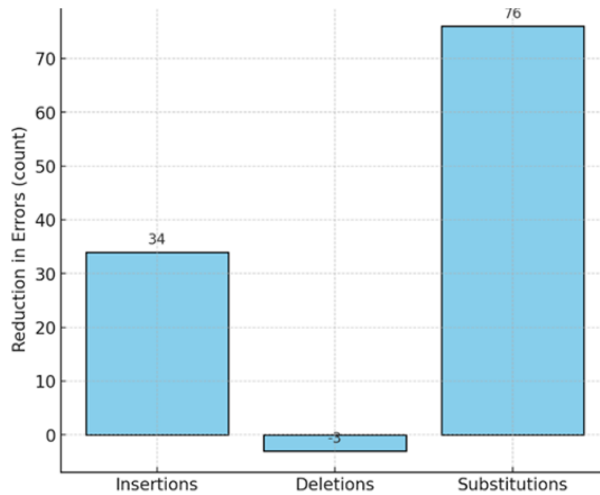


Figure 16. Error reduction statistics model 1-latest

4.4 Results on Synthetic Test Data

Results on synthetic test data show that fine-tuning the pretrained Arabic Kraken model with Kashmiri-specific data significantly reduces both CER and WER (Table 13). The final model achieves substantial gains in character-level accuracy, demonstrating the effectiveness of transfer learning and controlled augmentation for low-resource Kashmiri Nastaliq OCR.

Table 13. Optical Character Recognition (OCR) performance on synthetic Kashmiri Nastaliq test data

Model	Character Error Rate (CER)	Word Error Rate (WER)	Character Accuracy (%)
Pretrained Arabic Kraken (no finetuning)	0.45	0.95	55.00
Proposed Model (Initial)	0.45	0.95	54.91
Proposed Model (Fine-tuned - Final)	0.20	0.56	79.91

4.5 Results on Real-World Test Data

Performance on real-world data is lower than on synthetic test images due to noise, illumination variation, skew, and font inconsistencies present in scanned pages and mobile phone photographs (Table 14).

Nevertheless, the fine-tuned Kashmiri OCR model consistently outperforms the unadapted Arabic baseline, confirming its improved robustness in practical usage scenarios.

Table 14. Optical Character Recognition (OCR) performance on real-world Kashmiri printed text (scanned pages and mobile photos)

Model	Character Error Rate (CER)	Word Error Rate (WER)	Character Accuracy (%)
Pretrained Arabic Kraken (no finetuning)	0.52	0.98	48.00
Proposed Model (Fine-tuned - Final)	0.32	0.71	68.00

4.6 Baseline Comparison

The proposed Kashmiri OCR model is compared against two baselines:

- I. The pretrained Arabic Kraken model without fine-tuning, and
- II. A generic OCR baseline evaluated on the same test data.

This comparison provides context for the gains achieved through Kashmiri-specific fine-tuning (Table 15).

Table 15. Comparison with baseline Optical Character Recognition (OCR) models

Model	Test Data	Character Error Rate (CER)	Word Error Rate (WER)	Character Accuracy (%)
Generic OCR Baseline	Synthetic	0.49	0.96	51.00
Pretrained Arabic Kraken (no fine-tuning)	Synthetic	0.45	0.95	55.00
Proposed Kashmiri OCR (Fine-tuned)	Synthetic	0.20	0.56	79.91
Generic OCR Baseline	Real-World	0.55	0.99	45.00
Pretrained Arabic Kraken (no fine-tuning)	Real-World	0.52	0.98	48.00
Proposed Kashmiri OCR (Fine-tuned)	Real-World	0.32	0.71	68.00

The fine-tuned Kashmiri OCR model consistently outperforms both baseline systems on synthetic and real-world test data, demonstrating the effectiveness of Kashmiri-specific training while maintaining the baseline nature of the system.

4.7 Ablation Study

Ablation experiments were conducted to assess the contribution of key components in the proposed OCR pipeline, including transfer learning, data augmentation, image binarization, image trimming, and DPI/font-size variation. Each component was removed or modified independently while keeping all other settings unchanged, and the resulting impact on CER and WER was measured on the synthetic test set (Table 16).

Table 16. Ablation analysis of key Optical Character Recognition (OCR) pipeline components

Configuration	Character Error Rate (CER)	Word Error Rate (WER)
Full Model (All Components Enabled)	0.20	0.56
Without Transfer Learning	0.38	0.89
Without Data Augmentation	0.29	0.72
Without Image Binarization	0.25	0.65
Without Image Trimming	0.24	0.63
Fixed DPI/Font Size (No Variation)	0.27	0.69

The ablation results in above table indicate that transfer learning from a pretrained Arabic OCR model has the most significant impact on recognition performance, followed by data augmentation. Preprocessing steps such as binarization, trimming, and DPI/font-size variation also contribute to improved accuracy, though to a lesser extent.

These findings highlight the importance of combining transfer learning with controlled data preprocessing for OCR in low-resource scripts such as Kashmiri Nastaliq.

4.8 Character-Level Error Analysis

Table 17 presents the most frequent character confusions, primarily involving visually similar Nastaliq letters and diacritical marks.

Table 17. Most frequent character confusions in Kashmiri Nastaliq Optical Character Recognition (OCR)

Ground Truth Character	Predicted Character	Script Type	Error Type	Frequency
ب	پ	Arabic	Substitution	High
ن	پ	Arabic	Substitution	Medium
ن	ن	Kashmiri-specific	Substitution	Medium
ہ	ہ	Kashmiri-specific	Substitution	Medium
- (Fathah)	(omitted)	Diacritic	Deletion	High
- (Kasrah)	- (Fathah)	Diacritic	Substitution	Medium
- (Dammah)	(omitted)	Diacritic	Deletion	High
Space	(merged)	Whitespace	Deletion	Medium

Despite the targeted improvements, Kashmiri inherited diacritics remain the most challenging component for the OCR system. Their small visual footprint, contextual variability, and frequent overlap with base characters contribute to higher substitution and deletion rates compared to core Arabic script characters.

4.9 Inference Performance

Inference was conducted on an NVIDIA GPU-based system equipped with an NVIDIA RTX 3060 GPU, 12 GB GPU memory, and an Intel Core i7 CPU. The model processes printed Kashmiri text lines at an average rate of approximately 18–22 lines per second under batch inference settings.

All experiments were performed using Kraken version 3.5.2 with PyTorch backend. Hardware and software configurations are reported to facilitate reproducibility and practical deployment assessment.

5 Discussion

Beyond system accuracy, the analysis provides insight into the limitations of existing OCR engines when applied to Kashmiri Nastaliq. The observed error patterns, particularly for inherited diacritics and whitespace handling, reveal script-specific challenges that are not apparent in higher-resource Nastaliq languages such as Urdu. These findings offer practical guidance for dataset design, diacritic balancing, and post-processing strategies in future Kashmiri OCR research.

The performance gap observed between synthetic and real-world test data can be attributed to inherent differences in data characteristics. Synthetic images are generated using a limited set of fonts and controlled rendering parameters, which restricts font variability compared to real printed Kashmiri sources that exhibit diverse typography, print quality, and aging effects. Additionally, synthetic noise patterns introduced through augmentation do not fully capture real-world artifacts such as motion blur, uneven illumination, perspective distortion, and compression noise present in scanned pages and mobile phone photographs. Finally, the synthetic dataset operates at the line level and does not reflect layout complexity, including inter-line spacing variation, page margins, and background textures, which can indirectly affect segmentation and recognition in real documents. While synthetic data provides an effective foundation for training in low-resource settings, these factors collectively contribute to reduced performance on real-world inputs and highlight the need for richer real-world datasets and layout-aware extensions in future work.

5.1 Relation to Recent Deep Optical Character Recognition Approaches

Recent OCR research has increasingly explored transformer-based architectures and language-model-assisted recognition, which leverage long-range contextual modeling and implicit language understanding to improve recognition accuracy. Such approaches have demonstrated strong performance for high-resource languages and complex document layouts. However, their effective application typically requires large-scale annotated datasets, substantial computational resources, and extensive pretraining, which are currently unavailable for Kashmiri Nastaliq. In contrast, the present work prioritizes reproducibility and feasibility in a low-resource setting by adopting a CNN–RNN-based OCR engine with transfer learning. Nevertheless, transformer-based models and language-model-assisted post-processing represent promising directions for future research and may help address remaining challenges such as diacritic ambiguity and spacing errors once sufficient annotated data becomes available.

5.2 Practical Implications of High Word Error Rate

Although the proposed system achieves substantial improvements in character-level accuracy, the relatively high WER, primarily driven by spacing inconsistencies and diacritic-related errors, affects downstream usability. For text search and information retrieval, incorrect word boundaries can reduce recall and precision, as visually correct words may be tokenized incorrectly. In linguistic analysis, spacing and diacritic errors may impact morphological and phonological studies, where accurate vowel representation is critical. For archival digitization, high WER limits direct usability of OCR output for publication-quality text, necessitating human review or post-correction.

6 Conclusions

This research presented the design, implementation, and evaluation of an OCR system for Kashmiri text written in the Perso-Arabic (Nastaliq) script. Due to the linguistic complexity of Kashmiri and the limited availability of OCR resources, the work focused on developing a practical and extensible foundation rather than a fully developed commercial system.

Using careful dataset preparation, synthetic image generation, and transfer learning with the Kraken OCR framework, the system showed notable improvements in recognition accuracy over multiple training iterations. Significant reductions in character-level and word-level errors were observed, particularly for common and Arabic script characters, while Kashmiri-specific diacritics remain an area for further improvement.

This work establishes a reproducible baseline OCR framework for printed Kashmiri Nastaliq text. While substantial improvements are achieved through transfer learning and augmentation, diacritics and spacing remain key challenges. Future work will focus on expanding real-world annotated datasets, improving diacritic modeling, and integrating language-model-based post-processing.

Author Contributions

Conceptualization, S.A.F. and M.M.K.; methodology, S.A.F., M.M.K., and A.S.B.; software, M.M.K., A.S.B., and D.M.; validation, S.A.F., M.M.K., and M.Z.; formal analysis, S.A.F., A.S.B., and D.M.; investigation, M.M.K., A.S.B., and A.T.; resources, S.A.F. and M.Z.; data curation, M.M.K., D.M., and A.T.; writing—original draft preparation, S.A.F., M.M.K., and A.S.B.; writing—review and editing, S.A.F., M.Z., and A.T.; visualization, D.M. and A.T.; supervision, S.A.F. and M.Z.; project administration, S.A.F. All authors have read and agreed to the published version of the manuscript.

Data Availability

The data supporting the findings of this study are available from the corresponding author upon reasonable request.

The software tools and documentation used in this study are listed in references [21–27].

Conflicts of Interest

The authors declare no conflict of interest.

Declaration on the Use of Generative AI and AI-assisted Technologies

AI tools like Quilbot etc were used to assist with formatting and paraphrasing; all substantive analysis, interpretation, and conclusions were developed exclusively by the author.

References

- [1] B. B. Kachru, “The dying linguistic heritage of the Kashmiris: Kashmiri literary culture and language,” in *The Valley of Kashmir*, 2008, pp. 303–338.
- [2] O. N. Koul, “The Kashmiri language and society,” in *Kashmir and Its People: Studies in the Evolution of Kashmiri Society*, 2004, pp. 293–324.
- [3] S. Guha, “Empires, languages, and scripts in the Perso-Indian world,” *Comp. Stud. Soc. Hist.*, vol. 66, no. 2, pp. 443–469, 2024. <https://doi.org/10.1017/s0010417523000439>
- [4] M. B. Hall and L. J. Coin, “Pangenome databases improve host removal and mycobacteria classification from clinical metagenomic data,” *GigaScience*, vol. 13, p. giae010, 2024. <https://doi.org/10.1093/gigascience/giae010>
- [5] M. Humphries, L. C. Leddy, Q. Downton, M. Legace, J. McConnell, I. Murray, and E. Spence, “Unlocking the archives: Using large language models to transcribe handwritten historical documents,” *Hist. Methods*, vol. 58, no. 3, pp. 175–193, 2025. <https://doi.org/10.1080/01615440.2025.2500309>

- [6] S. M. U. Kumar, M. Azim, and S. M. K. Quadri, "Addressing the data gap: Building a parallel corpus for Kashmiri language," *Int. J. Inf. Technol.*, vol. 16, no. 7, pp. 4363–4379, 2024. <https://doi.org/10.1080/01615440.2025.2500309>
- [7] S. H. Nazki, "Tracing the path of the English language in Kashmir: A linguistic journey," *World J. Linguist. Lit.*, vol. 2, no. 1, pp. 1–7, 2025. <https://doi.org/10.61784/wjll3002>
- [8] M. A. Bhat, *The Changing Language Roles and Linguistic Identities of the Kashmiri Speech Community*. Newcastle upon Tyne, UK: Cambridge Scholars Publishing, 2017.
- [9] M. Bashir, V. Goyal, and K. J. Giri, "Challenges in recognition of Kashmiri script," in *Recent Innovations in Computing: Proceedings of ICRIC 2021*. Singapore: Springer, 2022, pp. 33–43.
- [10] S. Naz, A. I. Umar, R. Ahmad, S. B. Ahmed, S. H. Shirazi, I. Siddiqi, and M. I. Razzak, "Online cursive Urdu-Nastaliq script recognition using multidimensional recurrent neural networks," *Neurocomputing*, vol. 177, pp. 228–241, 2016. <https://doi.org/10.1016/j.neucom.2015.11.030>
- [11] S. A. Sattar, S. Haque, M. K. Pathan, and Q. Gee, "Implementation challenges for Nastaliq character recognition," in *International Multi Topic Conference*. Berlin, Heidelberg: Springer, 2008, pp. 279–285.
- [12] A. Rana and G. S. Lehal, "Offline Urdu OCR using ligature based segmentation for Nastaliq script," *Indian J. Sci. Technol.*, vol. 8, no. 35, pp. 1–9, 2015. <https://doi.org/10.17485/ijst/2015/v8i35/86807>
- [13] A. Kaur and G. S. Lehal, "A comprehensive survey of OCR for Devanagari script based languages," *Int. J. Res. Eng. Sci.*, vol. 9, no. 4, pp. 34–47, 2025. <https://doi.org/10.5281/zenodo.16277831>
- [14] Z. Jahan and S. Padmavathi, "Deciphering Kashmiri script: Challenges and advances in character recognition," in *2025 12th International Conference on Computing for Sustainable Global Development (INDIACom)*, Delhi, India, 2025, pp. 1–7.
- [15] F. Mushtaq, M. M. Misgar, M. Kumar, and S. S. Khurana, "UrduDeepNet: Offline handwritten Urdu character recognition using deep neural network," *Neural Comput. Appl.*, vol. 33, no. 22, pp. 15 229–15 252, 2021.
- [16] A. Naseer and K. Zafar, "Comparative analysis of raw images and meta feature based Urdu OCR using CNN and LSTM," *Int. J. Adv. Comput. Sci. Appl.*, vol. 9, no. 1, pp. 438–443, 2018. <https://doi.org/10.14569/ijacsa.2018.090157>
- [17] S. Shabbir and I. Siddiqi, "Optical character recognition system for Urdu words in Nastaliq font," *Int. J. Adv. Comput. Sci. Appl.*, vol. 7, no. 5, pp. 586–591, 2016. <https://doi.org/10.14569/ijacsa.2016.070575>
- [18] S. Arif and S. Farid, "From press to pixels: Evolving Urdu text recognition," *arXiv Preprints*, p. arXiv:2505.13943v2, 2025.
- [19] L. Ramamoorthy, N. Choudhary, and S. M. Bhat, "A gold standard Kashmiri raw text corpus," Central Institute of Indian Languages, 2019. <http://data.ldcil.org/a-gold-standard-kashmiri-raw-text-corpus>
- [20] P. B. K. Rajkhowa, "Deep learning model to revive Indian manuscripts," *Int. J. Sci. Res.*, vol. 12, no. 4, pp. 1365–1368, 2023. <https://doi.org/10.21275/sr23422084622>
- [21] Kraken OCR, "Kraken documentation," Kraken. <https://kraken.re/main/index.html>
- [22] B. Kiessling, "Training kraken," Kraken Documentation. <https://kraken.re/3.0/training.html>
- [23] Kraken OCR, "Kraken OCR annotation and transcription," Training Kraken. <https://kraken.re/5.3.0/training.html#annotation-and-transcription>
- [24] R. Smith, "Tesseract OCR documentation," GitHub. <https://github.com/tesseract-ocr/tesseract>
- [25] ImageMagick Studio LLC, "Mastering digital image alchemy," ImageMagick. <https://imagemagick.org/>
- [26] ImageMagick Studio LLC, "Command-line processing," ImageMagick. <https://imagemagick.org/script/command-line-processing.php>
- [27] Google, "Adding assets and images," Flutter Documentation. <https://docs.flutter.dev/ui/assets/assets-and-images>