Information and Knowledge Management (2025)
Clausius Scientific Press, Canada

DOI: 10.23977/infkm.2025.060209
ISSN 2523-5893 Vol. 6 Num. 2

Constructing a Machine-Learning-Ready Dataset for Low-Resource Fonts: A Case Study on the Uyghur Mongolian Script

Hongchen Chen[1,a], Chuhan Qiao[1,b], Kaitai Yang[2,c], Bilig Bato[1,d,*]

[1]College of Materials Science and Art Design, Inner Mongolia Agricultural University, Hohhot, Inner Mongolia Autonomous Region, China
[2]Makarov Marine Engineering Academy, Jiangsu Ocean University, Lianyungang, Jiangsu, China
[a]965157344@qq.com, [b]2944581552@qq.com, [c]1592243544@qq.com, [d]biligebatu@imau.edu.cn
**Corresponding author*

Keywords: Uyghur-Mongolian Script; Low-Resource Fonts; Glyph Extraction; Image Nor Malization; Dataset Construction; Deep Learning; OCR Preprocessing

Abstract: Progress in modern AI and machine learning depends on accessible, reproducible, and standardized datasets; however, data infrastructure for low-resource scripts remains limited. This study presents an end-to-end, fully reproducible pipeline that transforms Uyghur Mongolian font files into standardized, quality-controlled glyph datasets, accompanied by open protocols and tooling. A total of 13 Uyghur Mongolian fonts (Unicode U+1820–U+1842) were extracted, and five representative fonts were randomly selected for empirical evaluation. The proposed workflow achieved a 99.63% rendering success rate, with an average end-to-end processing time of approximately four seconds per font file. The resulting publication-grade glyph images are suitable for OCR benchmarking, digital typography, and document analysis. In addition, the pipeline supports SVG exports of native font outlines—preserving internal holes and contour hierarchies—for vector-level editing and stroke-aware style-transfer workflows. All procedures, configuration files, and scripts are fully documented to ensure one-click reproducibility; both code and data will be released under open licenses to facilitate reuse and extension. This dataset has also been validated through deep-feature analysis using a pretrained VGG19 model, confirming its structural consistency and suitability for machine-learning applications. By establishing a transparent, reusable, and machine-learning ready data foundation for an under-represented writing system, this work contributes to the digital preservation of cultural heritage and provides essential groundwork for future AI-based research on low-resource scripts.

1. Introduction

With the rapid advancement of digital technologies and artificial intelligence, demand for computational processing of minority and low-resource languages has grown substantially. The Uyghur Mongolian script, as a representative case, faces multiple challenges stemming from nonuniform font encoding, incomplete character coverage, and limited digital support. The Unicode Standard has provisioned official codepoints for traditional Mongolian script, e.g. “Mongolian” block (U+1800–U+18AF) and its supplement, facilitating more consistent encoding across platforms. However, empirical studies have documented that many existing OpenType fonts and rendering implementations remain incomplete or flawed, particularly in handling complex features such as positional variants, ligatures, and context-based shaping—underscoring the need for standardized, high-quality glyph resources[1].

The digitization of low-resource scripts is of fundamental importance to cultural preservation and sustainable development. These scripts are often tightly interwoven with specific ethnocultural traditions and thus serve as vessels of unique heritage. Experience shows that systematic digitization initiatives can facilitate the preservation and dissemination of traditional knowledge; academically, structured digital texts provide robust data for investigating language change, historical migration, and cross-cultural exchange. In efforts to revive historical movable type, prior work referencing the Kanjur font screening pipeline has underscored the necessity of a standardized selection–replication methodology for reconstructing historical glyph forms. Meanwhile, digital platforms expand the reach of minority scripts to broader audiences, thereby fostering intercultural understanding[2].

Recent studies on Traditional Mongolian indicate that direction-aware lightweight backbones can substantially improve layout analysis for vertically typeset text[3], while sequence-to-sequence (Seq2Seq) approaches with character-level language modeling effectively support online handwritten recognition[4].Together, these advances highlight the foundational value of standardized glyph inputs. Although a Mongolian online handwriting corpus exists (e.g., MOLHW, approximately 200 writers and 164,631 word samples)[5], curated corpora of standardized, printed-style glyphs for the Uyghur Mongolian script remain scarce, limiting training and evaluation for style transfer and high-accuracy OCR.

The Uyghur Mongolian script faces key challenges: fragmented encoding schemes, inconsistent font quality, poor compatibility, and a lack of standardized workflows and character inventories. Ligature rules and vertical layout further complicate processing, as many traditional algorithms are designed for horizontal text. Variations in resolution and contrast degrade recognition accuracy, while inter-font stylistic differences complicate model training.

High-quality, standardized, and aligned glyph images are essential for accurate style transfer in Uyghur Mongolian script. Without proper extraction and standardization, key stylistic features such as stroke thickness, trajectory, and connection patterns cannot be transferred.Existing font-processing tools fail to support the script's unique characteristics, leading to time-consuming, ad hoc solutions that hinder reproducibility and portability.

This study proposes a lightweight, automated workflow with a dual-interface tool (CLI + GUI) for Uyghur Mongolian font extraction and standardization. The tool ensures consistent output specifications and reusable processing protocols for subsequent OCR and style transfer tasks.

2. Materials and Methods

2.1 Study Objects and Character Set: Inclusion Criteria and Traceability

This study focuses on Uyghur-style Mongolian script fonts, including both vector (TTF, OTF) and bitmap (BDF) formats. The character encoding follows the Unicode Standard (v15.0), ensuring uniformity across different scripts. To eliminate bias, we fix the analysis set to the Unicode interval U+1820–U+1842, which includes 35 basic characters, and disable contextual joining rules and glyph substitutions to isolate the visual form of each glyph.

For traceability, we implement a dual-track mechanism, where a Python loop iterates over the 35 code points, ensuring each is processed without duplication or omission. The rendering process is standardized across all steps, with consistent canvas settings, anti-aliasing (4×), and supersampling. Each glyph is linked to its metadata, including identity, rendering details, timing, quality status, and file linkage.

Exceptions, such as missing glyphs or image corruption, are logged with detailed error codes and timestamps, ensuring transparency and preventing data bias. A strict one-to-one relationship is maintained between each character and its rendered glyph, allowing precise cross-font comparisons and ensuring data integrity.

2.2 Software Environment: Open-source Component Selection and Data Consistency Control

2.2.1 Component Selection and Version Pinning

For font rendering and rasterization, we adopt Pillow[6], which supports multi-format image output and fine-grained anti-aliasing, with a clean API and robust cross-platform behavior. For vector outline processing, we use FontTools[7] to parse TTF/OTF files and extract Bézier contours and control points for analysis. For vector-to-raster conversion, a Cairo-based SVG Rasterizer[8] is used for high-accuracy rasterization that preserves geometric detail at glyph edges—especially for complex strokes. For image processing, OpenCV[9] provides mature morphology, distance transforms[10], and edge detection, while NumPy[11] supports efficient array storage and matrix operations for pixel-level computations. For quality assessment, scikit-image provides SSIM and PSNR and other metrics with reliable implementations; for visualization, Matplotlib[12] produces publication-grade plots (e.g., boxplots, heatmaps) with high customizability.

2.2.2 Dual Image-representation Design

To enable both "texture evaluation" and "geometric feature analysis," we maintain two synchronized representations throughout the pipeline with distinct technical paths.

The grayscale representation is directly derived from the anti-aliased render, where pixel intensity (0–255) reflects local ink density. Semi-transparent edge pixels (e.g., gray value ≈ 128) represent anti-aliasing (AA) transition regions, preserving edge smoothness and fine texture. This representation is used for texture and edge smoothness evaluation, with SSIM from scikit-image quantifying similarity in texture and detail.

The binary ink-only representation is produced through three standardization steps. First, Otsu's [13]method thresholds the grayscale image to separate ink (0) from background (255). Second, polarity correction inverts the image if the background pixels exceed 50%, ensuring a standard “white background with black ink” format. Third, speckle suppression removes small components (default 5% of the largest connected ink region) using 8-connected component analysis, focusing on the core ink region.

2.3 Standardized Glyph-Export Pipeline: Dual Ingestion with a Single Normalization Path

2.3.1 Rendering Details and Consistency Constraints

In raster mode, the process follows three steps:

1. Allocate a high-resolution grayscale canvas (e.g., 128×128 with ×8 in CLI or ×6 in GUI).
2. Render the code point with Pillow's anti-aliasing, compute a tight bounding box (bbox), and center it on the canvas.
3. Standardize size and margins by tight-crop → INTER_AREA downsampling to 128×128 with

a margin ratio of 0.12, followed by centroid recentering. The grayscale output preserves sub-pixel edges for SSIM analysis, while a binary mask is derived using Otsu for geometry. This path ensures compatibility with the vector path' s normalization while reflecting screen anti-aliasing behavior.

In vector mode, the process also follows three stages:

1. Extract glyph paths and bounds with FontTools, then generate an SVG with a fixed transform (Y-axis flip + offset correction).
2. Render the SVG with CairoSVG into a high-resolution grayscale image (same target × render-scale as raster mode) with a white background.
3. Apply the same normalization—tight-crop → INTER_AREA downsampling to 128×128 with margin 0.12 → centroid recentering. Optionally, retain the SVG for audit or re-rendering. This ensures resolution-independent fidelity while making the pixels comparable to the raster path.

Two constraints ensure comparability across both modes:

1. Unify the temporary canvas size and supersampling factor to avoid resolution-related discrepancies.
2. Use the same interpolation scheme and target resolution for downsampling, ensuring identical per-pixel geometric constraints in both modes.

2.3.2 Normalization

Step 1: Tight crop and aspect-preserving scale — standardize size and margins.

We compute the minimal bounding rectangle on the binary ink-only image by scanning all non-zero pixels, then crop away empty borders. Given the margin ratio *m* (default 0.12), the maximal usable side length for the glyph on a 128×128 canvas is:

$$L_{usable} = 128 \times (1 - 2m) \tag{1}$$

The cropped glyph is scaled proportionally so that its longer side equals Lusable, ensuring no distortion, and is then placed at the canvas center with white padding, completing size and margin standardization.

Step 2: Binarization and polarity unification — standardize foreground/background polarity.

We apply Otsu's method to the grayscale image from Step 1 to obtain a binary image[8]. We then compute the proportions of black (ink) and white (background) pixels. If black > 50%, the image is deemed polarity-inverted (white text on black), and pixel values are inverted to enforce "white background with black ink."Step 3: Keep only the largest connected component — suppress speckles.

Using 8-connected component analysis, we identify all ink components, compute their areas, and retain only the largest one (core ink region). Regions below a relative threshold (default 5% of the largest component) are removed as noise (e.g., isolated pixels from anti-aliasing).

Step 4: Centroid relocation—standardize glyph position. On the binary image after retaining the largest component, we compute the geometric centroid via image moments[14]:

$$c_x = \frac{M_{10}}{M_{00}}, cy = \frac{M_{01}}{M_{00}} \tag{2}$$

Here,M_{00} is the area, and M_{10},M_{01} are the first-order moments.

The offsets from the canvas center (64, 64) are:

$$d_x = 64 - c_x, d_y = 64 - c_y \tag{3}$$

The ink region is translated by (d_x,d_y) so that its centroid coincides with the canvas center.

If a “keep-gray” mode is enabled (for later texture analysis), the same translation is also applied to the grayscale image to keep edges co-registered with the binary ink region.

Step 5: Boundary-contact correction — avoid ink touching the canvas border.

We scan the four borders (y=0,y=127,x=0,x=127) to detect any black pixels on the boundary. Formally:

$$\exists(x,y) \in \text{boundary}, I(x,y) = 0 \Rightarrow m \leftarrow + \Delta m \tag{4}$$

If contact is found, the margin ratio is incremented by a fixed step (e.g., from 0.12 to 0.14) and Steps 1–4 are re-run until contact disappears or a maximum margin (default 0.30) is reached (to prevent excessive downsizing). A maximum retry count (default 5) is also enforced.

Step 6: Direction correction (optional) — adapt to layout direction.

Uyghur-style Mongolian is used in both horizontal (left-to-right) and vertical (top-to-bottom) layouts. We infer a candidate direction from the bounding-box aspect ratio:

$$r = \frac{H}{W} \tag{5}$$

Where H is height and W is width. If r>1.5, the glyph is likely vertical; if r<0.67, it is likely horizontal. This inference is then combined with a user-specified preference (“horizontal-first” or “vertical-first”). If vertical layout is requested, ±90° rotations are applied using bilinear interpolation, and Step 4 (centroid relocation) is repeated to restore centering.

Step 7: Output and metadata packaging—integrate standardized images and metadata. We generate two standardized image products:

1. A binary ink-only image (mandatory), PNG lossless, single-channel 8-bit, values 0 for ink and 255 for background, 128×128 pixels, for geometric analysis (area, contour);
2. A grayscale image (optional), PNG lossless, single-channel 8-bit, values 0–255, preserving AA edge detail for texture evaluation (e.g., SSIM). For each image we produce a paired XML/JSON metadata file recording the Unicode code point (U+XXXX), character entity, rendering mode (raster/vector), parameters from each normalization step (margin ratio, rotation angle, whether “keep-gray” was applied), timestamp, batch ID, and quality status (normal/boundary-contact exception).
3. At the end of each batch, we automatically generate a “batch summary report” covering the number of successfully exported characters, counts/proportions of exceptions (render failures, boundary-contact), average per-glyph processing time, and resource usage (CPU, memory), supporting efficiency evaluation and quality control.

2.3.3 Scientific Validation

To verify the effectiveness of the standardization pipeline in removing non-stylistic factors, we define a protocol with multiple dimensions:

For geometric consistency, we select three fonts with distinct design styles (Font 1: traditional handwritten; Font 2: modern print; Font 3: simplified decorative) and render all 35 characters (U+1820–U+1842) both "before" and "after" standardization. We compare canvas occupancy (ink area / canvas area) and margin deviation (actual margin − 0.12). Before standardization, occupancy differences across fonts can reach 28% (Font 1 ≈ 68% vs. Font 3 ≈ 40%), and margin deviations range from −0.05 to 0.08. After standardization, occupancies converge to 76% ± 2%, with margin deviations shrinking to −0.01 to 0.01, indicating effective removal of size/margin confounds.

For feature stability, we extract core geometric features (stroke width, curvature, perimeter-to-area ratio) and compare variance before and after standardization. For example, cross-font variance in stroke width drops from 0.92 pixels (dominated by size differences) to 0.18 pixels after standardization, reflecting design-style differences (e.g., Font 1 mean width 1.9 px vs. Font 3 mean width 1.1 px).

For statistical testing, we apply paired t-tests (for approximately normal features like occupancy) and Wilcoxon signed-rank tests (for non-normal features like margin deviation) to assess pre/post differences. Results show significant changes ($p < 0.001$) with large effect sizes (|Cohen's d| > 1.2), supporting the effectiveness of the standardization.

For visualization, boxplots display distributional changes, histograms show margin deviation densities, and typical characters (e.g., U+1835 with long strokes, U+1822 with symmetric structure) are shown in "before vs. after" overlays with difference heatmaps (red for non-stylistic differences, blue for stylistic ones). These visualizations clearly show that non-stylistic biases (size/position) are reduced after standardization, while stylistic differences (stroke thickness, corner forms) are preserved.

2.4 Reference Rendering and Two-stage Alignment: Building a Robust Comparison Baseline

2.4.1 Design and Implementation of Reference Rendering

Reference images are generated with exactly the same parameters as in §2.3 (target 128×128, supersampling ×8, margin 0.12, bilinear interpolation, Otsu binarization, etc.). For example, the temporary canvas size and AA parameters match those used for the comparison images, ensuring equivalent rendering fidelity and edge detail. If any pipeline parameter (e.g., supersampling factor) is later adjusted, reference images must be regenerated accordingly to prevent parameter drift between reference and test images.

2.4.2 "Ink-first" Two-stage Alignment

Stage 1: Moment-based initialization (coarse alignment).

This stage leverages image moment features to remove large-scale differences between the test image (T) and the reference (R).Scale normalization via area ratio. The ink areas of both images are first computed from their zero-order moments (M00T and M00R). Because an object's area scales with the square of its linear dimension, the scaling factor between the two glyphs can be expressed as:

$$k = \sqrt{\frac{S_R}{S_T}} \tag{6}$$

Where ST and SR denote the ink areas of the test and reference glyphs. For instance, if T's ink area is 1.44× larger than R's, then k=0.83 , and T is proportionally downscaled to 83% of its original size. Bilinear interpolation is employed in this resizing step, as it preserves edge smoothness and reduces aliasing artifacts.Rotation correction via the principal inertia axis. After scale normalization, second-order central moments are used to estimate orientation. These moments are assembled into a covariance matrix:

$$C = \begin{bmatrix} \mu_{20} & \mu_{11} \\ \mu_{11} & \mu_{02} \end{bmatrix} \tag{7}$$

From this matrix, the principal axis orientation θ\thetaθ is derived from the eigenvector associated with the largest eigenvalue. Equivalently, it can be computed directly as:

$$\theta = \frac{1}{2}\operatorname{atan2}(2\mu_{11}, \ \mu_{20} - \mu_{02}) \tag{8}$$

The test glyph's orientation θTand the reference glyph's orientation θRare then compared, and the difference $\Delta\theta=\theta R-\theta T$ is applied as a corrective rotation. For example, if T is tilted 12° to the right while R is perfectly horizontal, then T is rotated by −12° to achieve alignment.Translation via centroid matching. Finally, the centroids of both glyphs are calculated from their first-order moments:

$$c_X = \frac{M_{10}}{M_{00}}, cy = \frac{M_{01}}{M_{00}} \tag{9}$$

The offsets from the canvas center are then computed as $\Delta x=C_x,R-C_x,T$ and $\Delta y=C_y,R-C_y,T$,and the test glyph is translated accordingly. This coarse stage is computationally efficient (typically requiring <10 ms per glyph) while removing large-scale misalignments such as millimeter-level shifts, rotations up to ±30°, and scale differences of nearly 2×. It establishes a robust initialization for the subsequent fine alignment stage.

Stage 2: Frequency-domain refinement plus local search (fine alignment).

Moments describe global geometry but may not ensure precise local edge alignment (e.g., stroke terminals or corner junctions) Sub-pixel shifts of only 0.2–0.8 px can still degrade metrics such as SSIM. The fine stage therefore combines phase correlation in the frequency domain with a localized spatial search.

Sub-pixel translation by phase correlation [15]. First, Canny edge extraction is applied to both the coarsely aligned test image T_1 and the reference image R [16]. The 2-D Fourier transforms of the resulting edge maps, denoted as $F_T(u,v)$ and $F_R(u,v)$, are then computed, and the normalized cross-power spectrum is obtained as:

$$P(u,v) = \frac{F_T(u,v) \cdot F_R^*(u,v)}{|F_T(u,v) \cdot F_R^*(u,v)|} \tag{10}$$

The inverse Fourier transform of P(u,v)P(u,v)P(u,v) yields the phase-correlation surface. The location of its peak corresponds to the estimated sub-pixel offset:

$$(\Delta x_{sub}, \Delta y_{sub}) = \arg\max P(u,v) \tag{11}$$

This offset is applied to obtain an updated test glyphT_2. A Hann window may optionally be used to reduce spectral leakage during this computation.Spatial local search maximizing IoU. Around T_2, a small search grid (±2 px with 0.2px steps) is constructed. For each candidate displacement, the Intersection-over-Union (IoU) between T_2 and R is computed on their binary ink masks (or optionally on dilated edge maps). The displacement that maximizes IoU is selected as the final translation, ensuring maximum local stroke overlap.Mirror validation and exception control. Since some glyphs may exist in mirrored forms (left–right or top–bottom), mirrored versions of T_2 are also tested. A mirrored candidate is accepted only if it improves IoU by more than a preset gain threshold (e.g., 5%). In addition, if the optimal displacement exceeds a cap (e.g., 3 px), the glyph is flagged as an "alignment exception" (e.g., due to extreme design divergence or damage) and is analyzed separately to avoid biasing aggregate metrics.

By combining sub-pixel precision in the frequency domain with spatial local optimization, this two-stage strategy reduces alignment error to within about 0.1 px, ties alignment quality directly to ink overlap (IoU), and avoids interference from grayscale transitions—thereby providing the positional consistency required for subsequent geometric comparisons and texture-based similarity measures, as shown in Figure 1.

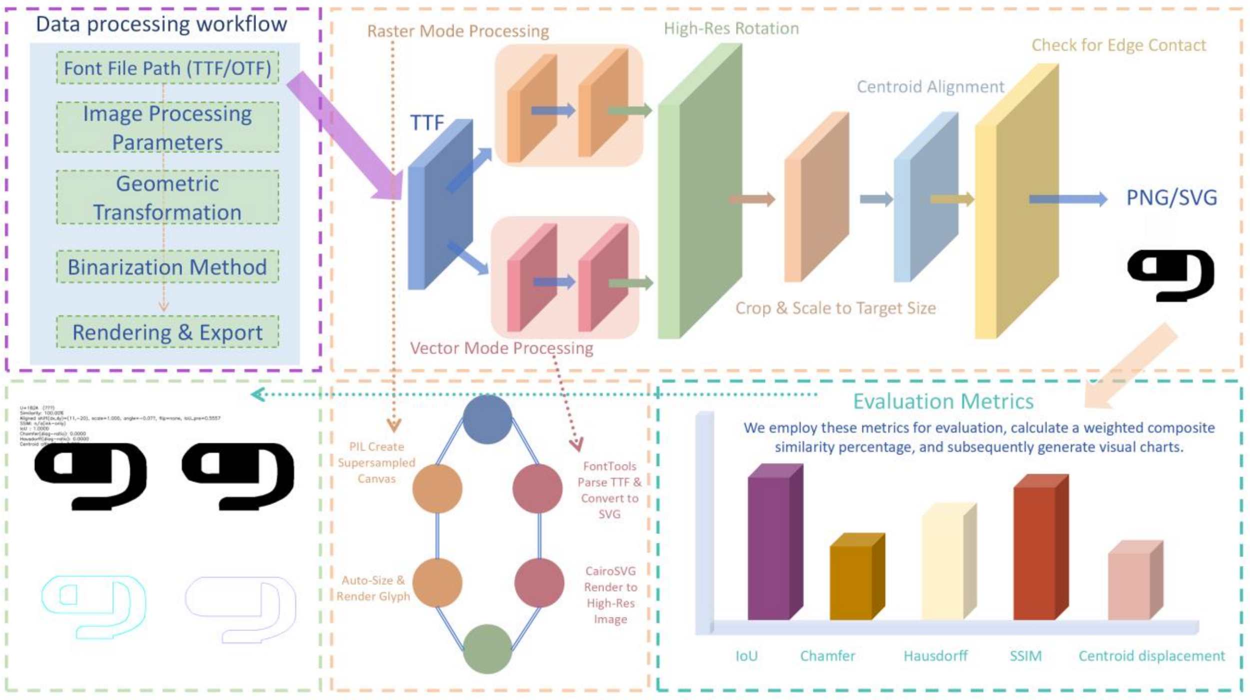


Figure 1: Font image extraction workflow, with similarity evaluated by IoU, Chamfer, Hausdorff, SSIM, and centroid offset.

2.5 Metric Suite

This study evaluates glyph consistency along three primary axes—structural overlap, edge shape, and geometric stability—with an optional grayscale structure term. Unless otherwise noted, distance metrics are used in symmetric form and normalized by the image diagonal length D, ensuring cross-resolution comparability.

Minimal notation:

X, Y ∈ {0,1}^{H×W}: binary ink images (reference, test).

E(X), E(Y): edge point sets extracted from X and Y (e.g., Canny).

A(S): pixel count (cardinality) of set S.

d(p, S) = min_{q ∈ S} ‖p − q‖₂: Euclidean distance from point p to set S.

D = √(H² + W²): image diagonal length (in pixels).

c(X), c(Y): ink centroids of X and Y (pixel coordinates).

(1) Structural overlap — Intersection over Union (IoU)[17]

Formula:

$$\mathrm{IoU} = \frac{A(X \cap Y)}{A(X \cup Y)} \tag{12}$$

Interpretation: primary positive indicator of structural agreement; sensitive to global shape and coverage.

(2) Edge shape — symmetric Chamfer distance (normalized)[18]

Pre-def.:

$$\mathrm{CD}(X \to Y) = \text{mean over } p \in E(X) \text{ of } d(p, E(Y)) \tag{13}$$

Main formula:

$$\text{SymChamfer} = \frac{[\text{CD}(X \to Y) + \text{CD}(Y \to X)]}{(2 \times D)} \tag{14}$$

Interpretation: captures average edge misalignment; complements IoU by emphasizing local boundary fidelity.

(3) Edge shape — symmetric Hausdorff distance (normalized)[19]

Pre-def.:

$$\text{HD}(X \to Y) = \text{max over } p \in E(X) \text{ of } d(p, E(Y)) \tag{15}$$

Main formula:

$$\text{SymHausdorff} = \frac{\max\{\text{HD}(X \to Y), \text{HD}(Y \to X)\}}{D} \tag{16}$$

Interpretation: captures worst-case edge deviation; sensitive to terminals, corners, and outliers.

(4) Geometric stability — centroid shift (pixels)

Formula:

$$\text{CentroidShift} = || c(X) - c(Y) ||_2 \tag{17}$$

Interpretation: first-line gate for residual translation or mass distribution mismatch.

(5) Optional grayscale structure — SSIM[20]

Formula:

$$\text{SSIM}(X, Y) = \frac{[(2\mu X \mu Y + C1)(2\sigma XY + C2)]}{[(\mu X^2 + \mu Y^2 + C1)(\sigma X^2 + \sigma Y^2 + C2)]} \tag{18}$$

Interpretation: enable when grayscale structure matters; disable for strictly ink-only pipelines.

2.6 Composite Similarity and Decision Criteria

We aggregate multi-source evidence into a single, interpretable composite similarity percentage (Sim%). A structure-first philosophy raises the influence of IoU (and SSIM, if enabled), while edge and geometric terms provide complementary constraints.

(1) Normalizing distances to scores (0–1)

Formula:

$$\text{Score}(d) = \max\{\frac{0, 1 - d}{\tau}\} \tag{19}$$

Where d ∈ { SymChamfer, SymHausdorff, CentroidShift } and$\tau$is the task-specific threshold for that metric.

Implementation tip: chooseτvia a validation quantile (e.g., 95th percentile) combined with application tolerance; D-normalization ensures resolution invariance.

(2) Linear aggregation to composite similarity

Formula:

$$\begin{aligned}\text{Sim\%} = 100 \times [\, & \text{wIoU}\cdot\text{IoU} + \text{wSSIM}\cdot\text{SSIM (optional)} + \text{wC}\cdot\text{Score(SymChamfer} \\ & + \text{wH}\cdot\text{Score(SymHausdorff)} + \text{wCe}\cdot\text{Score(CentroidShift)}\,]\end{aligned} \tag{20}$$

Suggested weights (sum to 1):

• Default (SSIM on): wIoU=0.50, wSSIM=0.20, wC=0.12, wH=0.12, wCe=0.06

• Ink-only (SSIM off): redistribute wSSIM proportionally to wIoU and (wC + wH)

(3) Decision thresholds and quality bands (tunable)

• Baseline acceptance: IoU ≥ 0.93; SymChamfer ≤ 0.02; CentroidShift ≤ 1.5 px; (if used) SSIM

≥ 0.97

• Research-grade: baseline + SymHausdorff at the same order as Chamfer (e.g., ≤ 0.02) and Sim% $\geq 98.5\%$

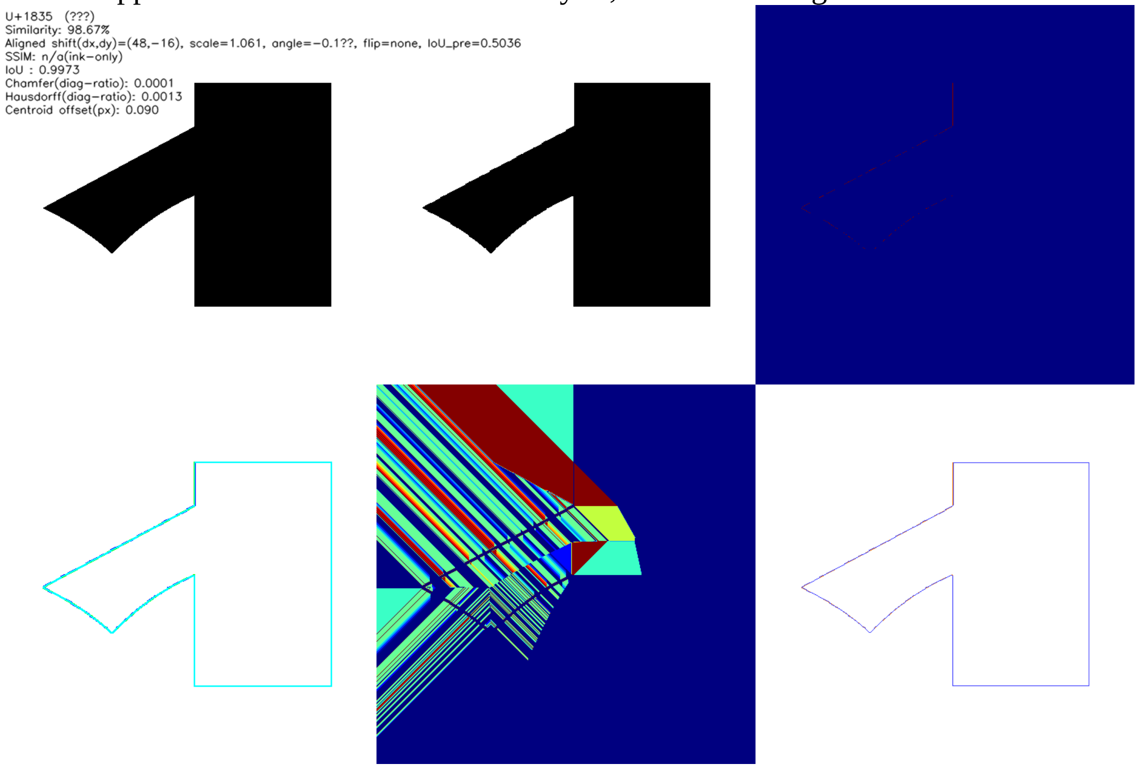
• Stringent/publishable: tighten τ and cutoffs (e.g., to 97.5th percentile on validation) so even worst-cases pass visual audit

(4) Robustness and interpretability

Chamfer (average error) + Hausdorff (extreme error) jointly stabilize edge assessments. Component-wise contributions (w $\times$ score) keep Sim% explainable; a per-glyph breakdown can be included in an appendix.

2.7 Visualization and Quality Control

The workflow also provides distributional summaries: histograms for IoU, Chamfer, Hausdorff, centroid shift, and Sim%, and scatter plots (e.g., IoU vs. Chamfer) to visualize correlations between structural overlap and edge deviation. Worst-case panel collages facilitate rapid auditing of tail behavior. A per-glyph metrics table (tabular format) and a compact web-style summary report are generated to support review and downstream analysis, as shown in Figure 2.



```
U+1835  (???)
Similarity: 98.67%
Aligned shift(dx,dy)=(48,-16), scale=1.061, angle=-0.1??, flip=none, IoU_pre=0.5036
SSIM: n/a(ink-only)
IoU : 0.9973
Chamfer(diag-ratio): 0.0001
Hausdorff(diag-ratio): 0.0013
Centroid offset(px): 0.090
```

Figure 2 Each glyph generates a one-page report, including reference/test ink images, alignment parameters, quantitative metrics and diagnostics.

Outputs include the necessary parameter settings and metric summaries to enable robust cross-platform re-evaluation. Distance metrics are symmetric and normalized by the image diagonal, ensuring resolution invariance. Ink-only mode is enabled by default; SSIM is computed only when the pipeline is not ink-only and the dependency is available, as shown in Figure 3, Figure 4 and Table 1.

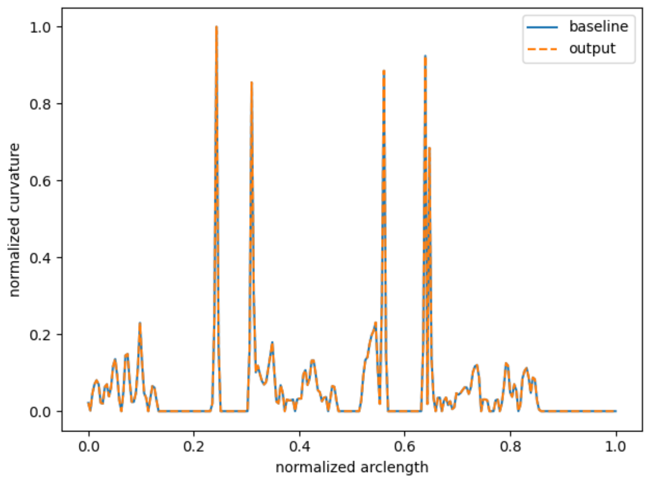


Figure 3 Boundary-curvature profiles (baseline vs output).

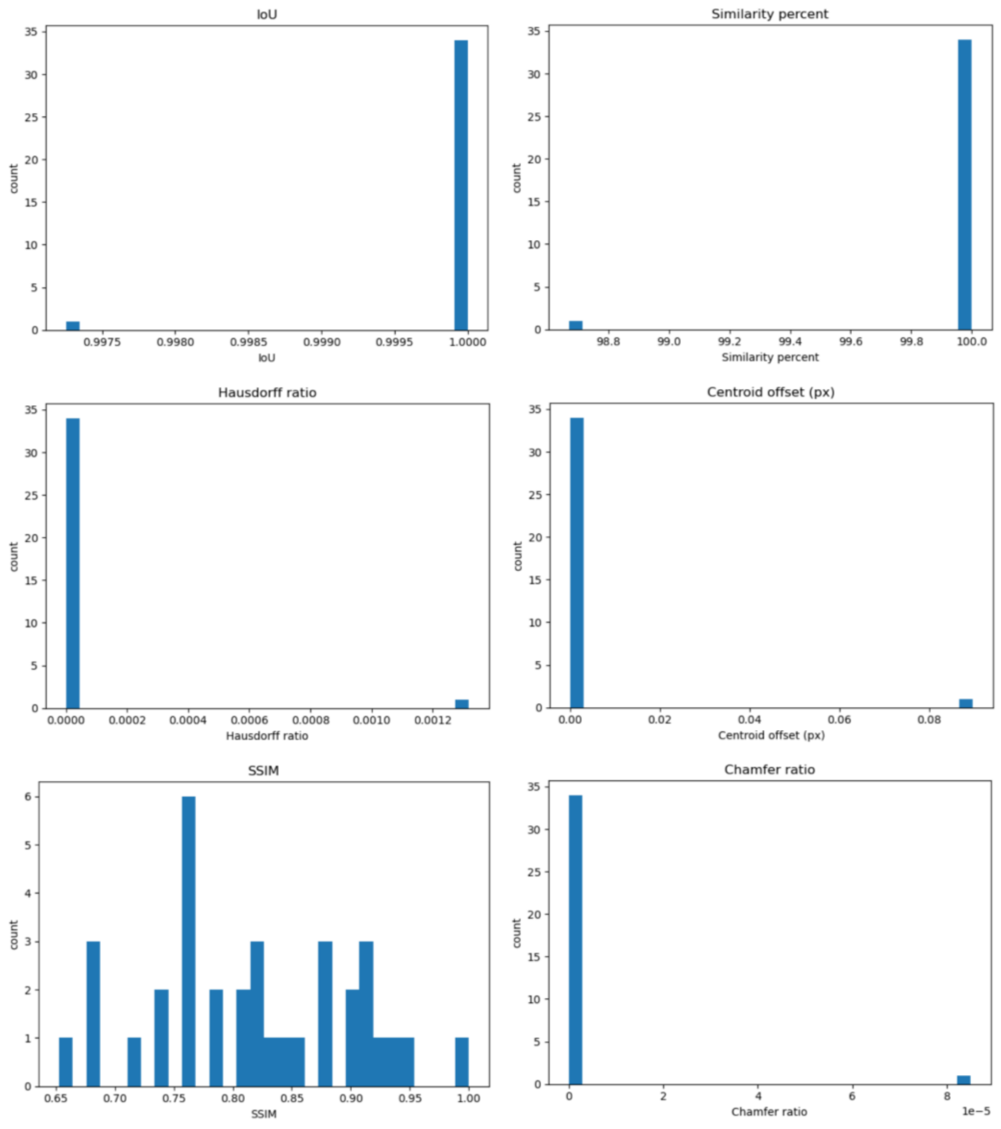


Figure 4 (a) IoU; (b) Similarity percent; (c) Hausdorff ratio; (d) Centroid offset (px) (e) SSIM; (f) Chamfer ratio;

Table 1 Intersection-over-Union (IoU) measures overlap (higher is better; max = 1), while Chamfer and Hausdorff are distance metrics (lower is better).

TTF	IoU	Chamfer	Hausdorff
A	1.0	0	0
B	0.991567	0.00015	0.000734
C	0.999921	0.000002	0.000038
D	0.998604	0.000025	0.000301
E	0.998991	0.000024	0.000301

2.8 Machine-Learning Readiness Evaluation (VGG19-based Validation)

To assess whether the standardized Uyghur Mongolian glyph dataset is suitable for AI-based analysis, a VGG19-based machine-learning readiness evaluation was performed. A pretrained VGG19 network was used to extract 512-dimensional embeddings for all 13 font families. The resulting deep features were analyzed using cosine distance metrics, principal-component analysis (PCA), and t-distributed stochastic neighbor embedding (t-SNE).

The intra-font distances concentrate around 0.1–0.2, whereas inter-font distances shift toward higher values (≈ 0.3–0.4). The mean intra-font distance (0.168) was smaller than the inter-font distance (0.200), yielding a positive separation gap (0.032). Although the silhouette coefficient was modest (−0.055), these results indicate that glyphs within each font remain geometrically consistent while cross-font pairs preserve discernible stylistic variation. The t-SNE visualization further demonstrates that the standardized glyphs form localized clusters per font family, confirming that the dataset occupies compact and separable regions in the deep-feature space, as shown in Figure 5 and Figure 6.

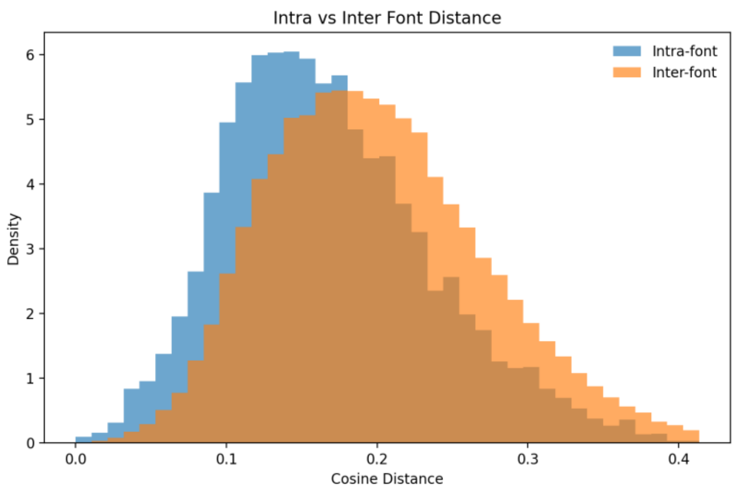


Figure 5 Distribution of intra-font (blue) and inter-font (orange) cosine distances computed from VGG19 embeddings. Lower intra-font variance and higher inter-font separation indicate structural consistency and stylistic diversity, confirming dataset stability and machine-learning readiness.

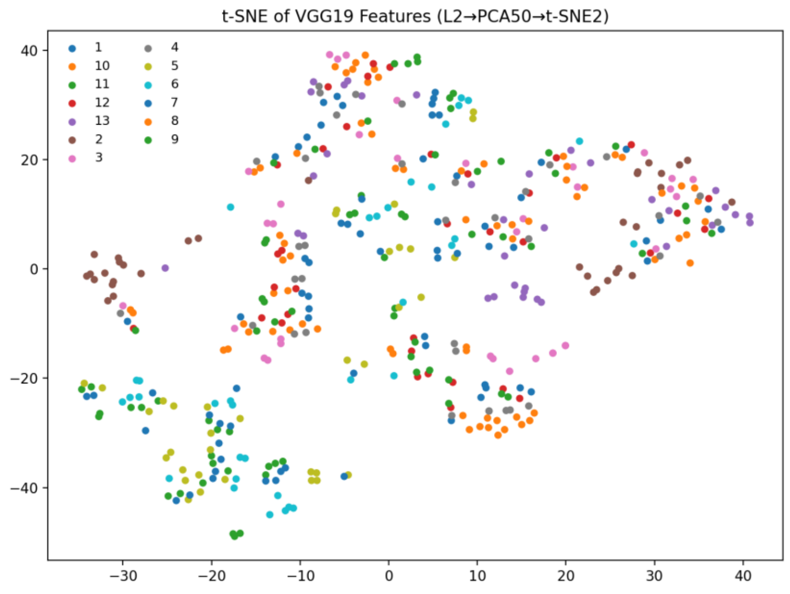


Figure 6 t-SNE visualization of the 512-dimensional VGG19 features (L2 → PCA50 → t-SNE2) for 13 standardized font families. Each color represents one font. Localized clusters demonstrate that the standardized glyphs occupy distinct, learnable regions in deep-feature space.

Overall, the VGG19-based analysis confirms that the standardized dataset possesses both structural consistency and stylistic diversity. These characteristics make it machine-learning–ready and suitable for downstream tasks such as font classification, clustering, and cross-font style transfer.

3. Results

Under unified rendering and geometric specifications, the standardized export and ink-first two-stage alignment produced publication-grade glyph sets for five Uyghur Mongolian font families (Unicode U+1820–U+1842). On our workstation, the end-to-end export averaged ~4 seconds per font set, improving iteration speed for batch curation, worst-case inspection, and parameter tuning. Repeatability is ensured by fixed operation order and batch-fixed parameters, yielding consistent outputs across re-runs.

Quality was assessed using Intersection-over-Union (IoU), symmetric Chamfer, symmetric Hausdorff distances (both normalized by the image diagonal), centroid shift, and optional SSIM (when the pipeline is not ink-only). Distributions concentrated within the predefined acceptance band (Methods): IoU ≥ 0.93, SymChamfer ≤ 0.02, CentroidShift ≤ 1.5 px, and SSIM ≥ 0.97 when computed. IoU captured global overlap, Chamfer reflected boundary displacement, Hausdorff highlighted extreme terminals/corners, and centroid shift flagged residual translation. Hausdorff was used for ranking and diagnostics, not as a hard cutoff, to avoid undue sensitivity to single-point outliers.

Outliers were mainly caused by boundary touching (insufficient margins), thin strokes near the binarization threshold, or contextual variants of the script. Visual inspection of the worst panels (prioritized by lowest IoU and largest Hausdorff) revealed that:

1. Adaptive margin enlargement restored clipped strokes, improving overlap without changing geometry.
2. Preserving edge grayscale (when not ink-only) mitigated thin-stroke erosion and stabilized edge distances.

3. Minor rotation (optional, 0.5 ° step) removed residual fringes in some cases, while the default ink-first alignment sufficed for most.
4. Mirroring was not enforced unless the IoU gain over the non-mirrored candidate was ≥ 0.12, preventing spurious flips while logging alternatives for traceability.

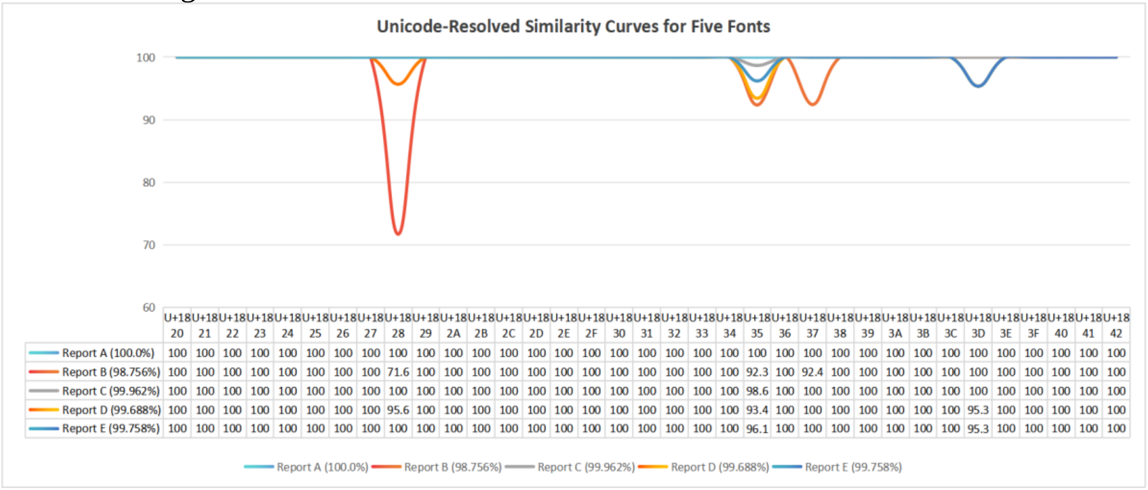
SVG tracing preserved inner holes and contour hierarchy while adhering to bitmap edges, supporting vector-level editing and compatibility with path-based style-transfer pipelines. The resulting black-on-white, centered 128×128 bitmaps and accurate SVGs were immediately usable for OCR pre-evaluation and cross-lingual style transfer.

Unified naming/metadata and per-glyph logs capture alignment parameters (translation, scale, rotation, mirroring), key metrics (IoU/Chamfer/Hausdorff/centroid; SSIM when available), pre-alignment soft-IoU, and the final Sim%. Batch summaries (means, quantiles, pass rates) enable apples-to-apples comparisons across fonts and parameter settings, and provide an audit trail for revisiting borderline samples, as shown in Table 2 and Figure 7.

Table 2 Accuracy is the proportion of glyphs meeting the acceptance criteria defined in Methods (higher is better).

TTF	Unicode	Accuracy
A	U1820~1824	100%
B	U1820~1824	98.756%
C	U1820~1824	99.962%
D	U1820~1824	99.688%
E	U1820~1824	99.758%

Notes. All runs used identical preprocessing and alignment settings; see Methods for the exact pass criteria and counting rules.



	U+1820	U+1821	U+1822	U+1823	U+1824	U+1825	U+1826	U+1827	U+1828	U+1829	U+182A	U+182B	U+182C	U+182D	U+182E	U+182F	U+1830	U+1831	U+1832	U+1833	U+1834	U+1835	U+1836	U+1837	U+1838	U+1839	U+183A	U+183B	U+183C	U+183D	U+183E	U+183F	U+1840	U+1841	U+1842
Report A (100.0%)	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Report B (98.756%)	100	100	100	100	100	100	100	100	71.6	100	100	100	100	100	100	100	100	100	100	100	100	92.3	100	92.4	100	100	100	100	100	100	100	100	100	100	100
Report C (99.962%)	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	98.6	100	100	100	100	100	100	100	100	100	100	100	100	100
Report D (99.688%)	100	100	100	100	100	100	100	100	95.6	100	100	100	100	100	100	100	100	100	100	100	100	93.4	100	100	100	100	100	100	100	95.3	100	100	100	100	100
Report E (99.758%)	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	96.1	100	100	100	100	100	100	100	95.3	100	100	100	100	100

Figure 7 Unicode-resolved similarity curves for five fonts (U+1820–U+1842).

4. Conclusions

We present a lightweight, dual-interface (CLI+GUI) glyph extraction and standardization toolchain for the low-resource Uyghur Mongolian script and validate it on five font families. Under unified rendering and geometric specifications, the ink-first two-stage alignment and multi-metric evaluation jointly ensure geometric consistency and interpretability; in practice, end-to-end export averages 4 seconds per set, markedly improving preparation efficiency and batch throughput. SVG vectorization adheres closely to bitmap edges and is immediately usable for vector-level editing and style-transfer pipelines. Unified naming, metadata, and per-glyph logs strengthen traceability and reproducibility. The tool reliably produces black-on-white, centered, and specification-consistent 128×128 bitmaps alongside high-fidelity vector outlines, providing dependable inputs for OCR pre-evaluation and cross-lingual/cross-font style transfer. Future extensions will target shaping-aware processing, higher resolutions with vector-native metrics, and learning-based fine registration, with broader validation across larger scales and additional scripts[21].

The VGG19-based analysis demonstrates that the standardized glyphs occupy learnable regions in a deep feature space, implying direct compatibility with convolutional and diffusion-based generative models. This confirms that the proposed dataset not only standardizes glyph geometry but also aligns with the representational assumptions of modern deep-learning systems.

References

[1] Batjargal B, Khaltarkhuu G, Kimura F, Maeda A. A Study of Traditional Mongolian Script Encodings and Rendering: Use of Unicode in OpenType Fonts[J]. International Journal on Asian Language Processing, 2011, 21(1): 23–43.

[2] Wurihan, Biligebatu. Standard Font Screening Method for Replicating Ancient Mongolian Kanjur Fonts[C]. Proceedings of the 2nd International Conference on Social Science, Public Health and Education (SSPHE 2018), 2019: 23–25.

[3] Zhou C, Ha M, Wu L. Direction-Aware Lightweight Framework for Traditional Mongolian Document Layout Analysis[J]. Applied Sciences, 2025, 15: 4594.

[4] Fan D, Sun Y, Wang Z, Peng Y. Online Mongolian Handwriting Recognition Based on Encoder–Decoder Structure with Language Model[J]. Electronics, 2023, 12: 4194.

[5] Pan Y, Fan D, Wu H, et al. A New Dataset for Mongolian Online Handwritten Recognition[J]. Scientific Reports, 2023, 13: 26.

[6] He R. Y., Kang S. H., Morel J.-M. A Formalization of Image Vectorization by Region Merging[J]. arXiv preprint, 2024.

[7] Zhu H.-K., Chong J. I., Hu T., Yi R., Lai Y.-K., Rosin P. L. SAMVG: A Multi-stage Image Vectorization Model with the Segment-Anything Model[J]. arXiv preprint, 2023.

[8] Liu Y.-T., Zhang Z., Guo Y.-C., Fisher M., Wang Z.-H., Zhang S.-H. DualVector: Unsupervised Vector Font Synthesis with Dual-Part Representation[J]. arXiv preprint, 2023.

[9] Bradski G. The OpenCV Library[J]. Dr. Dobb's Journal of Software Tools, 2000.

[10] Felzenszwalb P F, Huttenlocher D P. Distance Transforms of Sampled Functions[J]. Theory of Computing, 2012, 8: 415–428.

[11] Harris C R, et al. Array Programming with NumPy[J]. Nature, 2020, 585: 357–362.

[12] Hunter J D. Matplotlib: A 2D Graphics Environment[J]. Computing in Science & Engineering, 2007, 9(3): 90–95.

[13] Otsu N. A Threshold Selection Method from Gray-Level Histograms[J]. IEEE Transactions on Systems, Man, and Cybernetics, 1979, 9(1): 62–66.

[14] Hu M K. Visual Pattern Recognition by Moment Invariants[J]. IRE Transactions on Information Theory, 1962, 8(2): 179–187.

[15] Foroosh H, Zerubia J, Berthod M. Extension of Phase Correlation to Subpixel Registration[J]. IEEE Transactions on Image Processing, 2002, 11(3): 188–200.

[16] Canny J. A Computational Approach to Edge Detection[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1986, 8(6): 679–698.

[17] Rezatofighi H, et al. Generalized Intersection over Union: A Metric and a Loss for Bounding Box Regression[C]. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2019), 2019.

[18] Borgefors G. Hierarchical Chamfer Matching: A Parametric Edge Matching Algorithm[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1988, 10(6): 849–865.

[19] Huttenlocher D P, Klanderman G A, Rucklidge W J. Comparing Images Using the Hausdorff Distance[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1993, 15(9): 850–863.

[20] Wang Z, Bovik A C, Sheikh H R, Simoncelli E P. Image Quality Assessment: From Error Visibility to Structural Similarity[J]. IEEE Transactions on Image Processing, 2004, 13(4): 600–612.

[21] Li T M, Lukáč M, Gharbi M, Ragan-Kelley J. Differentiable Vector Graphics Rasterization for Editing and Learning[J]. ACM Transactions on Graphics (SIGGRAPH Asia), 2020, 39(6): Art. 193.