Supplementary Materials of NAMER: Non-Autoregressive Modeling for Handwritten Mathematical Expression Recognition

Chenyu Liu[1,2], Jia Pan[2], Jinshui Hu[2], Baocai Yin[2], Bing Yin[2], Mingjun Chen[2], Cong Liu[2], Jun Du[1†], and Qingfeng Liu[1,2]

[1] University of Science and Technology of China, Hefei, China
[2] iFLYTEK Research, Hefei, China
cyliu7@mail.ustc.edu.cn, jundu@ustc.edu.cn

1 NAMER Label Token Details

In this section, we provide details of labels in NAMER, including all local relation tokens and the attached imaginary tokens for them.

Table 1: Local Relation Tokens in VAT: Our defined imaginary relationship symbols are common to both the CROHME [1–3] dataset and the HME100K [4] dataset. Due to HME100K being a more complex dataset with a larger set of symbol classes, it contains a larger number of handwritten structural elements compared to CROHME.

	Dataset	Label Symbol
IRS	All	^ _ \limits
HSE	CROHME [1–3]	\frac \sqrt
HSE	HME100K [4]	\frac \sqrt \dot \ddot \boxed \widehat \overline \xlongequal \textcircled \xrightarrow \overrightarrow

† Corresponding author.

1.1 Local Relation Tokens in VAT

As mentioned in Section 3.2, local relation tokens in the Visual Aware Tokenizer (VAT) consists of Handwritten Structural Elements (HSE) and Imaginary Relationship Symbols (IRS). Details of HSE and IRS in different datasets are shown in Tab. 1.

1.2 Attached Imaginary Tokens in PGD

As outlined in Section 3.3, after obtaining all visible tokens $\mathbf{y}_{\mathrm{vat}}$ from VAT, the respective attached imaginary tokens related to the local relation tokens in $\mathbf{y}_{\mathrm{vat}}$ are added. For instance, in a mathematical expression $x^{y^z}+1$ with VAT results {“x”, “^”, “y”, “z”, “+”, “1”, }, the ending token “}” related to “^” must be added to fully recover the upper corner relation structure.

The definition of an attached imaginary token is straightforward: it represents the ending token for the local relation token, as illustrated in Tab. 2. It's important to note that most local relation tokens have a single “}” ending imaginary token, whereas “\frac” and “\sqrt” have two ending imaginary tokens. We have defined all attached tokens as “}”, ensuring uniformity with our label tokens in LaTeX.

Note that all the attached imaginary tokens are automatically generated by scanning all label strings using a simple Python script, not manually maintained. This script is written based on LaTeX grammar of special tokens.

Table 2: Details of all attached imaginary tokens. It is defined as the ending tokens for each local relation token.

Local Relation Token	Attached Imaginary Token
^	one '}' for ending
_	one '}' for ending
\limits	one '}' for ending
\dot	one '}' for ending
\ddot	one '}' for ending
\boxed	one '}' for ending
\widehat	one '}' for ending
\overline	one '}' for ending
\xlongequal	one '}' for ending
\textcircled	one '}' for ending
\xrightarrow	one '}' for ending
\overrightarrow	one '}' for ending
\frac	one '}' for numerator ending, one '}' for denominator ending
\sqrt	one '}' for radicand ending, one '}' for index ending

```python
# Python code for dynamic assignment in VAT. It's an example for one training sample.
# And it's easy to extend this code to support batch training.
import torch
from scipy.optimize import linear_sum_assignment

def vat_matching(P_vat, y_label, T_dwap, vocab_size, assign_kernel=(5, 5)):
    '''
    Parameters
    ----------------
    P_vat: FloatTensor, [K+1, H/8, W/8]
        predicted probabilities of VAT, which is the Eqn (2)'s \mathbf{P}
    y_label: LongTensor, [L, 1]
        all visible tokens, y_1, y_2, ..., y_L
    T_dwap: LongTensor, [L, 2]
        estimated spatial positions of y_label using a pretrained DWAP
    vocab_size: int
        vocabulary size
    assign_kernel: tuple (int, int)
        effective local matching window k_m * k_m for P_vat and T_vat

    Returns
    ----------------
    P_target: LongTensor, [H/8, W/8]
        The bipartite matching result of training target \mathbf{P}^* for VAT
    '''
    _, H, W = P_vat.shape  # get sizes of feature map
    L, _ = y_label.shape      # get length of visible tokens

    assign_pad = (assign_kernel[0]//2, assign_kernel[1]//2)  # compute paddings
    y_indices = torch.arange(L).cuda()  # a tensor for indexing

    # convert T_vat to a 3D matrix
    T_vat_mat = torch.zeros([L, H, W]).cuda()
    T_vat_mat[y_indices, T_dwap[y_indices, 0], T_dwap[y_indices, 1]] = 1.0

    # set effective matching positions using maxpooling
    T_vat_mat = torch.nn.functional.max_pool2d(T_vat_mat.unsqueeze(0), kernel_size=assign_kernel, stride=(1,1), padding=assign_pad)[0]

    # computing overall distance matrix for assigning H*W predictions to T tokens
    dist_mat = torch.zeros([L, H, W]).cuda()
    dist_mat[y_indices] = (P_vat[y_label[y_indices]] - T_vat_mat[y_indices]).abs()
    # limiting matching within a k_m x k_m window for each token
    dist_mat = dist_mat * T_vat_mat + (1 - T_vat_mat) * 1e6

    # hungarian algorithm for optimal assignment
    row_ind, col_ind = linear_sum_assignment(dist_mat.view(L, -1).cpu().numpy())
    h_ind = col_ind // W
    w_ind = col_ind % W

    # generating P_target for training VAT module
    P_target = (torch.ones([H, W]) * vocab_size).long() # initialized as all \varnothing.
    P_target[h_ind, w_ind] = y_label[row_ind]
    return P_target
```

Listing 1.1: Example Python Code for Dynamic Assignment Approach in VAT

2 Details of Dynamic Assignment Process

In this section, we present detailed code for the proposed bipartite matching-based dynamic assignment approach, which is initially introduced in Section 3.2 to generate training targets for VAT.

The code snippet in Listing 1.1 outlines the matching and assignment process. Specifically, given the estimated spatial positions $\mathbf{T} \in \mathbb{Z}^{L\times 2}$, for the parallel matching of all batches and tokens, we firstly convert $\mathbf{T}$ to a 3D matrix resembling an indicator function (line 34 in Listing 1.1). Subsequently, a max-pooling operation is applied to achieve local $k_m \times k_m$ window-based matching (line 37). Next, the overall distance matrix is computed between VAT's predicted probabilities $\mathbf{P}$ and the converted $\mathbf{T}_{mat}$. Finally, the Hungarian algorithm is employed for optimal assignment, resulting in the training target $\mathbf{P}^*$ for VAT.

3 Ablation Study on Path Selection

Path selection involves directed acyclic graph (DAG) construction and the longest path selection. We performed ablations on the edge E_{ij} of Eq. (10), finding a balanced 1:1 ratio of left-to-right (L2R) and right-to-left (R2L) score yields the optimal results, as shown in Tab. 3.

Table 3: Ablation study on edge value E_{ij} of DAG construction for Path Selection.

L2R:R2L	1:0	1:0.5	1:1	0.5:1	0:1
ExpRate	58.88	59.43	**60.51**	59.13	57.26

References

1. Mahdavi, M., Zanibbi, R., Mouchere, H., Viard-Gaudin, C., Garain, U.: Icdar 2019 crohme+ tfd: Competition on recognition of handwritten mathematical expressions and typeset formula detection. In: 2019 International Conference on Document Analysis and Recognition (ICDAR). pp. 1533–1538. IEEE (2019) 1
2. Mouchere, H., Viard-Gaudin, C., Zanibbi, R., Garain, U.: Icfhr 2014 competition on recognition of on-line handwritten mathematical expressions (crohme 2014). In: 2014 14th International Conference on Frontiers in Handwriting Recognition. pp. 791–796. IEEE (2014) 1
3. Mouchère, H., Viard-Gaudin, C., Zanibbi, R., Garain, U.: Icfhr2016 crohme: Competition on recognition of online handwritten mathematical expressions. In: 2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR). pp. 607–612. IEEE (2016) 1
4. Yuan, Y., Liu, X., Dikubab, W., Liu, H., Ji, Z., Wu, Z., Bai, X.: Syntax-aware network for handwritten mathematical expression recognition. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 4553–4562 (2022) 1