# Appendix

## A Postfix Transformation Conversion

Algorithm 1 is the pseudo-code for the convert the original feature transformation sequence $\Gamma$ to the postfix notation based sequence $\Upsilon$. In detail, we first initialize a list $\Upsilon$ and two stacks $S_1$ and $S_2$, respectively. For each element $\tau$ in $\Gamma$, we scan it from left to right. When getting a feature ID token, we push it to $S_2$. When receiving a left parenthesis, we push it to $S_1$. When obtaining any operations, we pop each element in $S_1$ and push them into $S_2$ until the last component of $S_1$ is the left bracket. Then we push this operation into $S_1$ When getting a right parenthesis, we pop every element from $S_1$ and then push into $S_2$ until we confront a left bracket. Then we remove this left bracket from the top of $S_1$. When the end of the input $\tau$ encounters, we append every token from $S_2$ into $\Upsilon$. If this $\tau$ is not the last element in $\Gamma$, we will append a <SEP> token to indicate the end of this $\tau$. After we process every element in $\Gamma$, we add <SOS> and <EOS> tokens to the beginning and end of $\Upsilon$ to form the postfix notation-based transformation sequence $\Upsilon = [\gamma_1, \cdots, \gamma_M]$. Each element in $\Upsilon$ is a feature ID token, operation token, or three other special tokens. We convert each transformation sequence through this algorithm and construct the training set with them.

**Algorithm 1:** Postfix transformation sequence conversion

**input :** Feature transformation sequence $\Gamma$
**output :** Postfix notation based transformation sequence $\Upsilon$

$\Upsilon \longleftarrow \emptyset$;
**for** $\tau \leftarrow \Gamma$ **do**
  $S_1, S_2 \longleftarrow \emptyset, \emptyset$;
  **for** $\gamma \leftarrow \tau$ **do**
    **if** $\gamma$ **is** left bracket **then**
      $S_1.push(\gamma)$;
    **else if** $\gamma$ **is** right bracket **then**
      **while** $t \leftarrow S_1.pop()$ **is not** left bracket **do**
        $S_2.push(t)$;
    **else if** $\gamma$ **is** operation **then**
      **while** $S_1$*.peek()* **is not** left bracket **do**
        $S_2.push(S_1.pop())$;
      $S_1.push(\gamma)$;
    **else**
      $S_2.push(\gamma)$;
  **while** $S_2$**not** $= \emptyset$ **do**
    $\Upsilon.append(S_2.pop(0))$;
  **if** $\tau$ **is not** *the last element* **then**
    $\Upsilon.append($<SEP>$)$;
Add <SOS> and <EOS> to the head and tail of $\Upsilon$ respectively;
**return** $\Upsilon$

## B Experimental Settings and Reproducibility

### B.1 Hyperparameter Settings and Reproducibility

The operation set consists of *square root, square, cosine, sine, tangent, exp, cube, log, reciprocal, quantile transformer, min-max scale, sigmoid, plus, subtract, multiply, divide*. For the data collection part, we ran the RL-based data collector for 512 epochs to collect a large amount of feature transformation-accuracy pairs. For the data augmentation part, we randomly shuffled each transformation sequence 12 times to increase data diversity and volume. We adopted a single-layer LSTM

Table 3: Time complexity comparison between MOAT and DIFER

| Dataset | Model Name | Data Collection 512 instances | Model Training | Solution Searching | Performance |
|---|---|---|---|---|---|
| Wine Red | MOAT | 921.6 | 5779.2 | 101.2 | 0.559 |
| Wine White | MOAT | 2764.8 | 7079.6 | 103.4 | 0.536 |
| Openml_618 | MOAT | 4556.8 | 30786.1 | 111.3 | 0.692 |
| Openml_589 | MOAT | 4044.8 | 8942.3 | 105.2 | 0.656 |
| Wine Red | DIFER | 323.2 | 11 | 180 | 0.476 |
| Wine White | DIFER | 1315.8 | 33 | 624 | 0.507 |
| Openml_618 | DIFER | 942.3 | 33 | 534 | 0.408 |
| Openml_589 | DIFER | 732.5 | 157 | 535 | 0.463 |

Table 4: Space complexity comparison on MOAT with different dataset

| Dataset | Sample Number | Column Number | Parameter Size |
|---|---|---|---|
| Airfoil | 1503 | 5 | 139969 |
| Amazon employee | 32769 | 9 | 138232 |
| ap_omentum_ovary | 275 | 10936 | 155795 |
| german_credit | 1001 | 24 | 141127 |
| higgs | 50000 | 28 | 141899 |
| housing boston | 506 | 13 | 139004 |
| ionosphere | 351 | 34 | 143057 |
| lymphography | 148 | 18 | 139969 |
| messidor_features | 1150 | 19 | 140162 |
| openml_620 | 1000 | 25 | 141320 |
| pima_indian | 768 | 8 | 138039 |
| spambase | 4601 | 57 | 147496 |
| spectf | 267 | 44 | 144987 |
| svmguide3 | 1243 | 21 | 140548 |
| uci_credit_card | 30000 | 25 | 141127 |
| wine_red | 999 | 12 | 138618 |
| wine_white | 4900 | 12 | 138618 |
| openml_586 | 1000 | 25 | 141320 |
| openml_589 | 1000 | 25 | 141320 |
| openml_607 | 1000 | 50 | 146145 |
| openml_616 | 500 | 50 | 146145 |
| openml_618 | 1000 | 50 | 146145 |
| openml_637 | 500 | 50 | 146145 |

as the encoder and decoder backbones and utilized 3-layer feed-forward networks to implement the predictor. The hidden state sizes of the encoder, decoder and predictor are 64, 64, and 200, respectively. The embedding size of each feature ID token and operation token was set to 32. To train MOAT, we set the batch size as 1024, the learning rate as 0.001, and $\lambda$ as 0.95 respectively. For inferring new transformation sequences, we used top-20 records as the seeds with beam size 5.

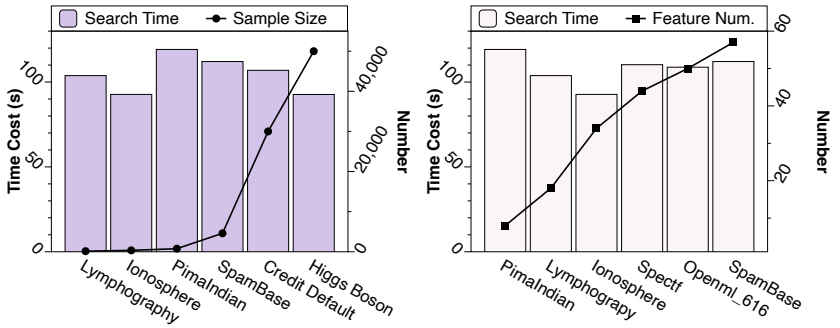
### B.2 Experimental Platform Information

All experiments were conducted on the Ubuntu 18.04.6 LTS operating system, AMD EPYC 7742 CPU, and 8 NVIDIA A100 GPUs, with the framework of Python 3.9.10 and PyTorch 1.8.1.

## C Experimental Results

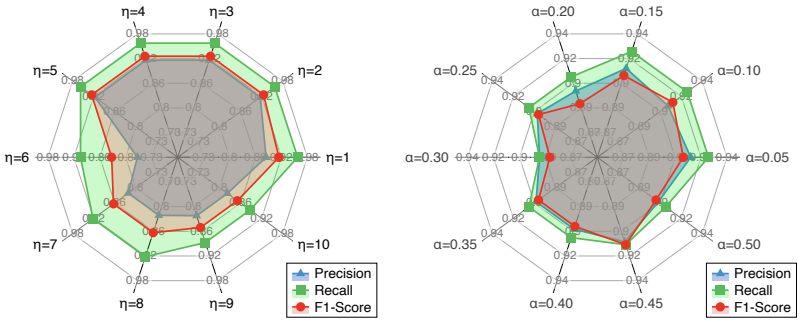
### C.1 Time complexity analysis.

To analyze the time complexity of MOAT, we selected the SOTA model DIFER as a comparison baseline. Figure 3 shows the time costs of data collection, model training, and solution searching in terms of seconds. We let both MOAT and DIFER collect 512 instances in the data collection phase. We found that the increased time costs for MOAT occur in the data collection and model training phases. However, once the model converges, MOAT 's inference time is significantly reduced. The underlying driver is that the RL-based collector spends more time gathering high-quality data, and the




(a) Sample Size (b) Feature Number

Figure 6: Scalability check of MOAT in search time based on sample size and feature number.




(a) Search Step Size $\eta$ (under $\alpha = 0.05$) (b) Training Trade-off $\alpha$ (under $\eta = 1$)

Figure 7: Parameter sensitivity on search step size $\eta$ and trade-off parameter $\alpha$ on Spectf dataset.

sequence formulation for the entire feature space increases the learning time cost for the sequential model. But, during inference, MOAT outputs the entire feature transformation at once, whereas DIFER requires multiple reconstruction iterations based on the generated feature space's dimension. The less inference time makes MOAT more practical and suitable for real-world scenarios.

### C.2 Space Complexity Analysis.

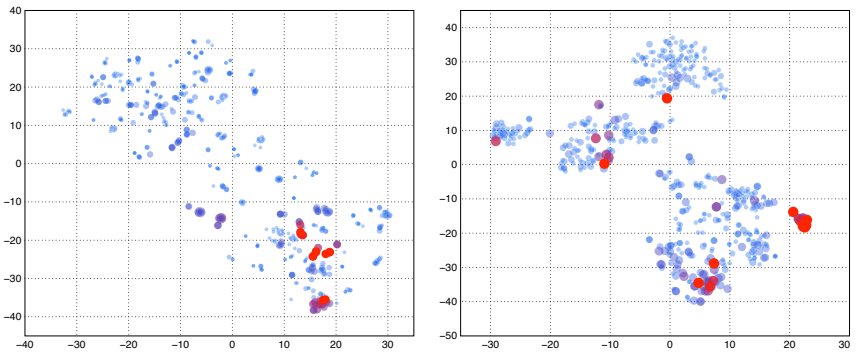
To analyze the space complexity of MOAT, we illustrate the parameter size of MOAT when confronted with different datasets. Table 4 shows the comparison results. We can find that the model size of MOAT keeps relatively stable without significant fluctuations. The underlying driver is that the encoder-evaluator-decoder learning paradigm can embed the knowledge of discrete sequences with variant lengths into a fixed-length embedding vector. Thus, such an embedding process can make the parameter size to be stable instead of increasing as the data size grows. Thus, this experiment indicates that MOAT has good scalability when confronted with different scaled datasets.

### C.3 Scalability Check

We visualized the changing trend of the time cost of searching for better feature spaces over sample size and feature dimensions of different datasets. Figure 6 shows the comparison results. We found that the time cost of MOAT keeps stable with the increase of sample size of the feature set. A possible reason is that MOAT only focuses on the decision-making benefits of feature ID and operation tokens instead of the information of the entire feature set, making the searching process sample size irrelevant. Another interesting observation is that the search time is still stable although the feature dimension of the feature set varies significantly. A possible explanation is that we map transformation records of varying lengths into a continuous space with a constant length. The searching time in this space is input dimensionality-agnostic. Thus, this experiment shows the MOAT has excellent scalability.

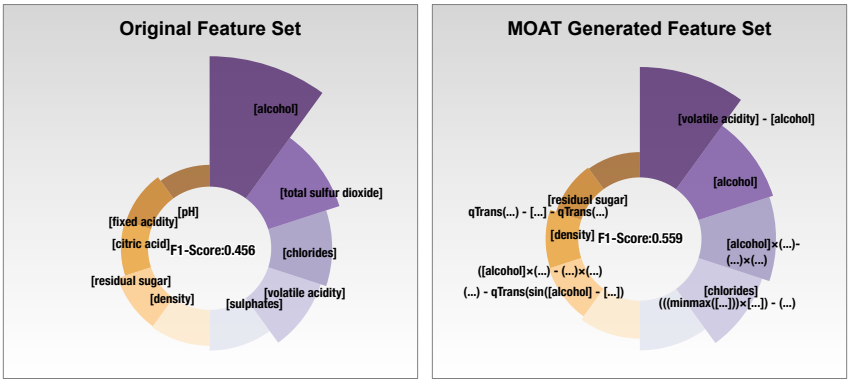
### C.4 Parameter sensitivity analysis

To validate the parameter sensitivity of the search step size $\eta$ (See section 3.5) and the trade-off parameter $\alpha$ in the training loss (See section 3.4), we set the value of $\eta$ from 1 to 10, and set the value of $\alpha$ from 0.05 to 0.50 to observe the difference. Figure 7 shows the comparison results in terms of precision, recall, and F1-score. When the search step size grows, the downstream ML performance initially improves, then declines marginally. A possible reason for this observation is that a too-large



(a) Openml_616 (b) Airfoil

Figure 8: The visualization of learned transformation sequence embedding (from MOAT).




(a) Original Feature Space (b) MOAT Generated Feature Space

Figure 9: Comparison of traceability on the original feature space and the MOATgenerated one.

step size may make the gradient-ascent search algorithm greatly vibrate in the continuous space, leading to missing the optimal embedding point and transformed feature space. Another interesting observation is that the standard deviation of the model performance is lower than 0.01 under different parameter settings. This observation indicates that MOAT is not sensitive to distinct parameter settings. Thus, the learning and searching process of MOAT is robust and stable.

### C.5 Learned embedding Analysis.

We selected Airfoil and Openml_616 as examples to visualize their learned continuous embedding space. In detail, we first collected the latent embeddings generated by the transformation records. Then, we use T-SNE to map them into a 2-dimensional space for visualization. Figure 8 shows the visualization results, in which each point represents a unique feature transformation sequence. The size of each point means its downstream performance. The bigger point size indicates that the downstream performance is superior. We colored the top 20 embedding points according to the performance in red. We found that the distribution locations of the top 20 embedding points are different. A potential reason is that the corresponding transformation sequences of the top 20 embedding points are different lengths. The sequence reconstruction loss distributes them to different areas of the embedding space. Moreover, we observed that the top 20 embedding points are close in the space even though the positions are different. The underlying driver is that the estimation loss makes these points with good performance clustered. Thus, this case study reflects that the reconstruction loss and estimation loss make the continuous space associate the transformation sequence and the corresponding model performance.

### C.6 Traceability case study

We selected the top 10 essential features for prediction in the original, and MOAT transformed feature space of the Wine Quality Red dataset for comparison. Figure 9 shows the comparison results. The texts associated with each pie chart are the corresponding feature name. The larger the pie area is, the more critical the feature is. We found that almost 70% critical features in the new feature space are generated by MOAT and they improve the downstream ML performance by 22.6%. This observation

indicates that MOAT really comprehends the properties of the feature set and ML models in order to produce a more effective feature space. Another interesting finding is that '[alcohol]' is the essential feature in the original feature set. But MOAT generates more mathematically composited features using '[alcohol]'. This observation reflects that MOAT not only can capture the significant features but also produce more effective knowledge for enhancing the model performance. Such composited features can make domain experts trace their ancestor resources and summarize new analysis rules for evaluating the quality of red wine.

## D Compared MOATwith SOTAs

Recent studies tried to convert the feature transformation into a continuous optimization task to search the optimal feature space efficiently. *DIFER* [8] is a cutting-edge method that is comparable to our work in problem formulation. However, the following constraints limit its practicality: 1) DIFER collects transformation-accuracy data at random, resulting in many invalid training data with inconsistent transformation performances; 2) DIFER embeds and reconstructs each transformed feature separately and, thus, ignores feature-feature interactions; 3) DIFER needs to manually decide the number of generated features, making the reconstruction process ad-hoc. 4) the greedy search for transformation reconstruction in DIFER leads to suboptimal transformation results. To fill these gaps, we first implement an RL-based data collector to automate high-quality transformation record collection. We then leverage the postfix expression idea to represent the entire transformation operation sequence to model feature interactions and automatically identify the number of reconstructed features. Moreover, we employ beam search to advance the robustness, quality, and validity of transformation operation sequence reconstruction.