Two-Stage Feature Generation with Transformer and Reinforcement Learning

Wanfu Gao[1,2], **Zengyao Man**[1,2], **Zebin He**[1,2], **Yuhao Tang**[1,2], **Jun Gao**[1,2*] and **Kunpeng Liu**[3]
[1]College of Computer Science and Technology, Jilin University, China
[2]Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, China
[3]Department of Computer Science, Portland State University, Portland, OR 97201 USA
gaowf@jlu.edu.cn, manzy23@mails.jlu.edu.cn, hezb2121@mails.jlu.edu.cn, tangyh2121@mails.jlu.edu.cn, gaocheng23@mails.jlu.edu.cn, kunpeng@pdx.edu

Abstract

Feature generation is a critical step in machine learning, aiming to enhance model performance by capturing complex relationships within the data and generating meaningful new features. Traditional feature generation methods heavily rely on domain expertise and manual intervention, making the process labor-intensive and challenging to adapt to different scenarios. Although automated feature generation techniques address these issues to some extent, they often face challenges such as feature redundancy, inefficiency in feature space exploration, and limited adaptability to diverse datasets and tasks. To address these problems, we propose a Two-Stage Feature Generation (TSFG) framework, which integrates a Transformer-based encoder-decoder architecture with Proximal Policy Optimization (PPO). The encoder-decoder model in TSFG leverages the Transformer's self-attention mechanism to efficiently represent and transform features, capturing complex dependencies within the data. PPO further enhances TSFG by dynamically adjusting the feature generation strategy based on task-specific feedback, optimizing the process for improved performance and adaptability. TSFG dynamically generates high-quality feature sets, significantly improving the predictive performance of machine learning models. Experimental results demonstrate that TSFG outperforms existing state-of-the-art methods in terms of feature quality and adaptability.

1 Introduction

Feature generation is a critical aspect of the construction of high-performance machine learning models. High-quality features can enhance the robustness, generalization capability, and interpretability of models [Zheng and Casari, 2018; Dong and Liu, 2018]. For instance, as Figure 1 shows, combining weight and height features into a Body Mass Index (BMI) [Obese, 1998] feature can significantly improve a model's ability to predict health conditions. As the scale and complexity of data increase, traditional feature engineering methods often struggle to uncover the complex patterns and underlying relationships within the data. Consequently, automatic feature generation techniques, especially those based on deep learning, have become a focal point of research [He *et al.*, 2021]. Recently, reinforcement learning has also provided new ideas for feature generation [Liu *et al.*, 2019; Wang *et al.*, 2022]. Despite their ability to extract complex features, deep learning methods still face challenges in adapting to different tasks and achieving efficient optimization [LeCun *et al.*, 2015]. This study introduces a two-stage feature generation framework that integrates pre-training with reinforcement learning-based fine-tuning to achieve more efficient and precise feature generation.

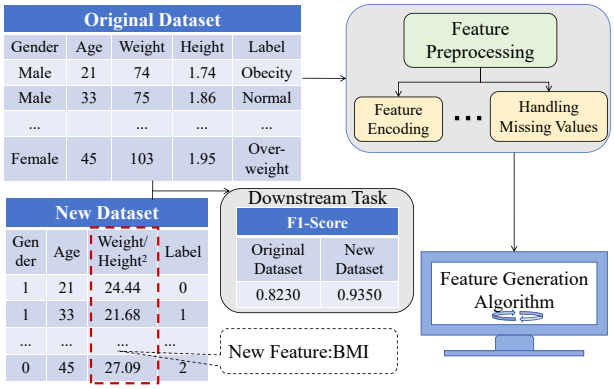


Figure 1: The original dataset is preprocessed and then fed into the feature generation algorithm. This algorithm creates new features and produces an enhanced dataset. The enhanced dataset is used for downstream tasks. Results show that the F1 score of the enhanced dataset is better than the original dataset.

The novelty of this framework lies in its staged training method, which enables stepwise optimization of feature generation through pre-training and reinforcement learning fine-tuning. In the first stage of pre-training, we employ an Encoder-Decoder architecture [Sutskever, 2014], optimizing the model using cross-entropy loss or mean squared error loss. The encoder extracts latent information from the raw dataset, while the decoder reconstructs this information into logically coherent sequences of feature combinations. For example, initial features

*Corresponding author

might appear as $[V1, \text{EOS}, V2, \text{EOS}, V3, \text{STOP}]$, and during decoding, more intricate feature combinations like $[+V1, +V2, \text{EOS}, -V3, \times V1, \text{STOP}]$ are generated. The goal of this stage is to build a stable initial model for feature generation.

Upon completing the first stage of pre-training, the second stage incorporates the Proximal Policy Optimization (PPO) algorithm [Schulman *et al.*, 2017] from reinforcement learning to fine-tune the model. This phase dynamically adjusts the model weights through a reward mechanism tied to specific downstream tasks, ensuring that the generated features better align with the requirements of these tasks. Specifically, the reward value of the generated feature combinations will be calculated. Based on the reward signal, the model optimizes its generation strategy, progressively learning high-quality feature representations. The PPO algorithm excels at balancing the magnitude of updates, preventing issues such as gradient explosion during the training process.

This two-stage training strategy offers several advantages. In the first stage, the Encoder-Decoder structure generates features step by step and computes the loss at each step, ensuring the stability and consistency of the generated features. In the second stage, the introduction of the PPO algorithm endows the feature generation process with dynamic optimization capabilities, enhancing task adaptability.

In summary, this paper introduces a novel feature generation framework, which leverages an encoder-decoder architecture integrated with Proximal Policy Optimization (PPO). The proposed framework enhances feature generation by dynamically adjusting the optimization strategy and improving adaptability across different tasks. Using a two-stage training process, we address the challenges of feature redundancy and inefficient exploration in traditional methods. Our main contributions include:

- Encoder-Decoder Architecture for Efficient Feature Generation: We propose an encoder-decoder architecture that effectively models the feature generation process, allowing for the extraction of meaningful latent representations from raw data. This ensures high-quality feature combinations for downstream tasks.
- Two-Stage Training with PPO Fine-Tuning: We introduce a two-stage method where pre-training initializes the model, and reinforcement learning fine-tuning optimizes feature generation based on specific task requirements. This allows the framework to dynamically adjust its strategy and enhance task performance.
- Enhanced Adaptability and Efficiency: By incorporating reinforcement learning and task-specific feedback, our framework can efficiently generate features tailored to diverse datasets and models, offering better performance and computational efficiency compared to existing methods.

2 Related Work

Automated feature generation plays a critical role in enhancing the performance of machine learning models [Domingos, 2012]. Common methods in this field can generally be divided into expansion-reduction methods and search-based methods, both aiming to generate meaningful and high-quality features for predictive modeling.

Expansion-reduction methods generate a large pool of candidate features through various transformations and then reduce redundancy by selecting the most relevant subset. For example, ExploreKit [Katz *et al.*, 2016] follows a three-step workflow: candidate feature generation, ranking, and evaluation. It applies transformation functions to the entire dataset and evaluates the performance of the predictive model to select the most valuable features. Similarly, FEADIS [Dor and Reich, 2012] generates new features by randomly combining original features with mathematical functions, while Autofeat [Horn *et al.*, 2020] expands the feature space using nonlinear transformations before selecting a small subset of features for inclusion. OpenFE [Zhang *et al.*, 2023] is an efficient expansion-reduction method that combines FeatureBoost and a two-stage pruning algorithm, enabling it to rapidly and accurately identify useful new features on large-scale datasets.

Despite their contributions, these methods often face challenges related to the computational burden and redundancy caused by the exponential growth of candidate features. As the number of features increases, selecting truly relevant ones becomes increasingly difficult, This prompts researchers to develop alternative approaches that enhance representation completeness [Wang *et al.*, 2021].

Search-based methods adopt a more targeted method by using search algorithms to explore potential feature transformations and combinations. These methods have shown potential in overcoming the limitations of expansion-reduction techniques. For instance, GRFG [Wang *et al.*, 2022] utilizes group reinforcement learning to select operations for groups of features and performs operations between them, effectively managing feature dependencies. TransGraph [Khurana *et al.*, 2018] uses a transformation graph and Q-learning algorithm to generate higher-order features. DIFER [Zhu *et al.*, 2022] introduces a feature optimizer within an encoder-predictor-decoder structure that maps features to a continuous vector space, optimizes embeddings along the gradient direction induced by the predictor, and recovers better features from the optimized embeddings via the decoder.

In addition, some studies on multi-label feature selection have proposed relevance metrics and feature decomposition strategies to ensure task-specific feature quality while reducing redundancy [Zhang *et al.*, 2021; Zhang and Gao, 2021; Li *et al.*, 2024].

In summary, while expansion-reduction methods can create a vast array of potential features, search-based methods offer a more strategic exploration of the feature space [Komer *et al.*, 2014], potentially leading to more effective and efficient feature engineering.

3 The Proposed Method

3.1 Problem Settings

Given a dataset $\mathcal{D} = \{\mathcal{F}, y\}$, where $\mathcal{F}$ represents the original feature set and y represents the target label set. The feature set $\mathcal{F} = \{V_1, V_2, \ldots, V_N\}$ consists of N discrete features.

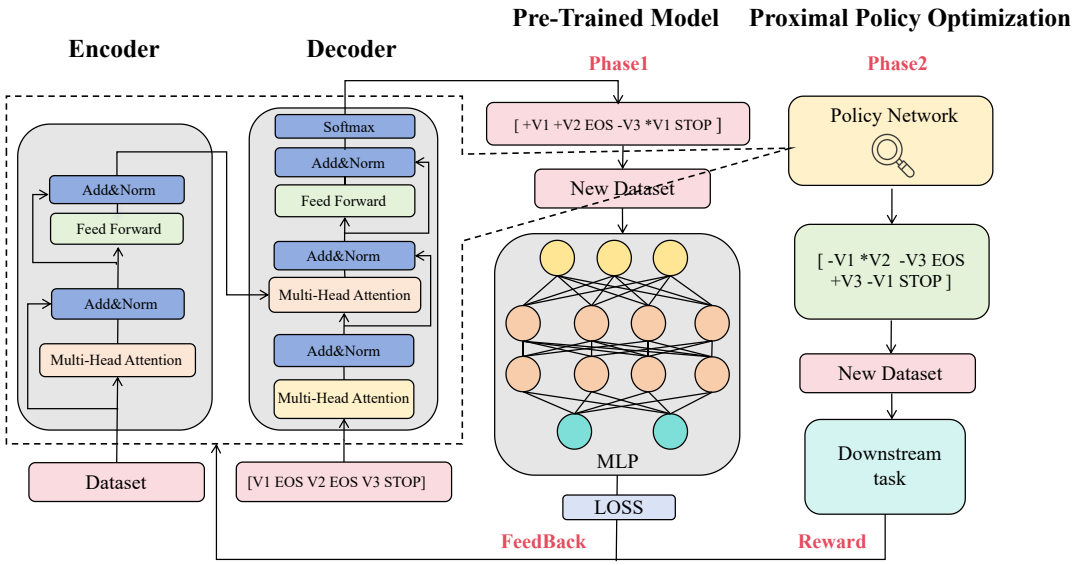


Figure 2: Overview of TSFG. In the first phase, the encoder-decoder model is optimized by calculating the loss. In the second phase, the encoder-decoder model serves as a policy network using Proximal Policy Optimization (PPO) to fine-tune the model based on reward feedback. This optimizes feature generation to adapt to specific downstream tasks, ensuring that feature generation can efficiently adapt to a variety of task requirements.

We aim to find the optimal feature set $\mathcal{F}^*$ that maximizes the performance metric of downstream tasks, defined as:

$$\mathcal{F}^* = \arg\max_{\mathcal{F}'} M_A(\mathcal{F}', y), \tag{1}$$

where A represents a downstream machine learning model (e.g., random forests, SVM, or neural networks), and M denotes the evaluation metric, such as accuracy, *F1-score*, or mean squared error.

For continuous features, mathematical operations include "absolute value", "square", "inverse", "logarithm", "square root", "cube", "addition", "subtraction", "multiplication", and "division". For discrete features, mathematical operations include "cross" [Luo *et al.*, 2019] and "addition".

3.2 Overall Framework

The framework combines reinforcement learning, an encoder-decoder architecture, and a feature transformation mechanism, aiming to generate optimized features to address challenges such as avoiding redundant feature generation and optimizing feature transformation sequences. The workflow is based on an encoder-decoder architecture as the main structure, a pre-training phase for initialization, and a reinforcement learning phase for fine-tuning.

At the core of the framework is an encoder-decoder architecture that can efficiently explore the feature space and generate new features. We select the Transformer architecture [Vaswani *et al.*, 2017] as the encoder-decoder model due to its superior capability in handling sequential problems [Raffel *et al.*, 2020]. Its self-attention mechanism effectively captures long-range dependencies and complex patterns within the data [Bahdanau, 2014]. The encoder processes the input dataset, mapping it into a latent representation that captures the structure and interdependencies of the original features. This latent representation serves as the foundation from which the decoder generates sequences of feature transformations.

The decoder is responsible for generating feature transformation sequences, which are produced step-by-step through an autoregressive mechanism [Graves, 2013a]. The output of each step depends on the result of the previous step, ensuring that the feature transformation process proceeds coherently and effectively captures the dependencies among features. To avoid generating invalid features, operations are combined with features to form a unified token. For example, operations like $+V1$, $-V2$, etc., are treated as tokens. The sequence output by the decoder is similar to $[+V1, +V2, \text{EOS}, -V3, \times V1, \text{STOP}]$, where EOS represents the end-of-feature token and STOP represents the end-of-sequence token. In this way, we generate two new features: $V1 + V2$ and $-V3 \times V1$, thereby expanding the feature set and improving the data representation for downstream machine learning tasks.

The pre-training phase focuses on initializing the encoder-decoder model. The encoder extracts latent representations of the dataset, and the decoder generates feature transformation sequences based on these representations. These transformations generate an augmented feature set, which is optimized by evaluating its performance on a pre-trained MLP model. The optimization objective is to minimize the cross-entropy loss or mean squared error loss, thereby enhancing the quality of feature generation and its adaptability to tasks.

To encourage exploration during training, a sampling mechanism with temperature scaling is adopted [Guo *et al.*, 2017]. This mechanism adjusts the sampling probabilities of certain transformations, enabling the model to explore a

wide range of feature combinations. The pre-trained encoder-decoder provides a solid foundation for the reinforcement learning phase, accelerating convergence and enhancing performance.

After pre-training, the model is fine-tuned using the Proximal Policy Optimization (PPO) algorithm. This phase focuses on iteratively improving the feature generation process to maximize cumulative rewards. The reward function is based on improvements in the performance of the downstream task, such as increased classification accuracy or F1 score. In each iteration of the PPO phase, similar to the pre-training stage, the encoder generates a latent representation of the dataset, and the decoder proposes candidate feature transformations based on this representation. These transformations are evaluated using external performance metrics, and a reward is calculated for the generated sequence. The PPO algorithm adjusts the parameters of the encoder-decoder to maximize the expected reward, balancing exploration and exploitation.

The PPO fine-tuning enables the framework to dynamically adapt to the data, generating task-optimized feature sets. This method not only ensures that the generated features enhance performance but also maintains their interpretability and relevance to the problem domain.

3.3 Pre-Training Phase for Encoder-Decoder Model

The goal of the pre-training phase is to enable the encoder-decoder model to learn an initial strategy. A new feature set is generated by the transformation sequence, which is then processed by the pre-trained MLP to compute predictions and calculate the prediction loss based on the true labels. By minimizing the prediction loss between the generated transformation sequences and the true labels, the model learns how to enhance the representations of features. The encoder-decoder is implemented based on the Transformer architecture.

Input: The input for the pre-training phase includes a dataset $D_{\text{train}} = \{F, y\}$, where F represents the original feature set, and y is the target variable. Additionally, a validation dataset D_{val} is used to evaluate the model's performance and generalization ability during training.

Output: The output is a pre-trained encoder-decoder model capable of generating feature transformation sequences for creating new features.

The encoder maps the input features F into a latent space $\mathcal{Z}$, where the latent representation $z \in \mathcal{Z}$ captures the structural relationships among the features. This ensures that the decoder has a comprehensive understanding of the data, allowing it to extract hierarchical feature representations. Formally, this process is expressed as:

$$z = \text{Encoder}(F; \theta_e), \tag{2}$$

where θ_e represents the parameters of the encoder. The latent space Z effectively captures the complex relationships between features, providing a compact yet informative representation for subsequent transformation.

The decoder then receives the latent representation z and predicts the transformation operation a, which is applied to the features. This process can be formalized as:

$$a = \text{Decoder}(z; \theta_d), \tag{3}$$

where θ_d represents the parameters of the decoder.

Furthermore, the decoder uses a temperature parameter T to adjust the probability distribution of the output sequence [Graves, 2013b].

$$P(a, z) = \frac{\exp(a/T)}{\sum_i \exp(a_i/T)}, \tag{4}$$

where T is the temperature parameter, a_i represents one of all possible actions. The temperature parameter controls the smoothness of the output probability distribution. When the temperature T is high, the generated features become more random, leading to a more exploratory method. Conversely, when T is low, the model tends to select features with higher probabilities, resulting in more deterministic outputs.

In this process, the decoder generates new feature transformation operations. These operations are applied to the original training dataset D_{train}, creating a new dataset $D_{\text{train}}^{\text{new}}$. The dataset $D_{\text{train}}^{\text{new}}$ is then fed into a pre-trained MLP, which outputs the predicted probability distribution for each sample's class. Finally, the loss between these predicted probabilities and the true labels is calculated to evaluate the quality of the generated feature transformations. The parameters of the encoder-decoder model are adjusted through backpropagation to minimize the loss function. The framework inputs the data into the MLP to compute the loss after generating a complete feature (i.e., when the EOS token is generated), rather than waiting to generate the entire transformation sequence. This step-by-step evaluation provides timely feedback on the effectiveness of each generated feature, thereby promoting faster convergence during training and further improving the quality of the generated features.

For classification tasks, the cross-entropy loss is defined as:

$$L_{\text{classification}} = -\frac{1}{N} \sum_{i=1}^{N} \left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right], \tag{5}$$

where N is the number of training samples. y_i is the true label of the i-th training sample. $\hat{y}_i$ is the predicted probability for the i-th sample, obtained by the pre-trained MLP after applying the current generated features to the training dataset D_{train}.

For regression tasks, the mean squared error (MSE) loss is defined as:

$$L_{\text{regression}} = \frac{1}{N} \sum_{j=1}^{N} (\hat{y}_j - y_j)^2, \tag{6}$$

where N is the number of training samples. $\hat{y}_j$ is the predicted value of the j-th feature. y_j is the true value of the target feature for the j-th sample.

3.4 Proximal Policy Optimization (PPO)

The Proximal Policy Optimization (PPO) fine-tuning phase is designed to refine the feature generation process by leveraging reinforcement learning. While the pre-training phase

provides a strong initialization for the encoder-decoder architecture, the PPO phase ensures that the model continues to optimize feature transformations based on feedback from downstream tasks. This phase is critical for dynamically adapting to specific datasets and tasks. The final new dataset is generated at this stage.

The objective of the PPO fine-tuning phase is to iteratively improve the feature generation process by maximizing a reward function that reflects the performance improvement of downstream tasks. Unlike the pre-training phase, where training is performed after generating each feature, in the PPO phase, the model parameters are optimized based on feedback from downstream tasks after generating the complete transformation sequence. This method comprehensively evaluates the overall performance of the complete feature set and the synergistic relationships among features, ensuring that the optimization direction aligns more closely with the requirements of downstream tasks, thereby producing higher-quality feature sets with greater global coherence.

The feature generation process is modeled as a Markov Decision Process (MDP). The encoder-decoder serves as the policy network, with the state, action, and reward defined as follows:

State (s): The current state represents the feature set.

Action (a): An action corresponds to a transformation operation selected by the decoder. For example, $+V_1$, $-V_2$, $*V_2$.

Reward (r): The reward is the performance improvement of the new dataset in the downstream task. The reward can be expressed as:

$$r = \Delta\text{Metric} = \text{Metric}(\mathcal{D}_{\text{new}}) - \text{Metric}(\mathcal{D}), \tag{7}$$

where $\mathcal{D}_{\text{new}}$ is the dataset with the newly generated feature set.

Policy (π): The policy determines which action to apply, given the current state. PPO is used to iteratively optimize the policy to maximize cumulative rewards.

The PPO fine-tuning workflow begins with initializing the encoder-decoder model using the weights obtained during the pre-training phase. This ensures that the model starts fine-tuning with a well policy for feature generation.

At each iteration, the encoder generates latent representations of the current dataset, which are passed to the decoder. The decoder samples candidate transformation sequences based on the current policy $\pi_\theta(a|s)$. The sequence is applied to the original dataset to generate a new dataset. The updated dataset is evaluated using a downstream machine learning model (e.g., a classifier or regressor). The model calculates the sequence's reward based on the performance improvement achieved by the new dataset.

Using the computed rewards, PPO adjusts the policy parameters θ to maximize the expected cumulative reward. The policy objective is defined as:

$$L_{\text{PPO}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right], \tag{8}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio, $\hat{A}_t$ is the advantage function, and ϵ is a clipping parameter to limit the magnitude of policy updates. To encourage the exploration of diverse transformations, an entropy regularization term is added to the PPO objective [Cai *et al.*, 2023]. This term prevents the policy from converging prematurely to suboptimal solutions by promoting randomness in action selection. The process of sampling, evaluation, and policy updates is repeated for a predefined number of iterations or until convergence criteria are met.

Algorithm 1 Encoder-Decoder Training with PPO Fine-Tuning

Input: Dataset $\mathcal{D} = \{\mathcal{F}, y\}$, where $\mathcal{F} = \{V_1, V_2, \ldots, V_N\}$ is the feature set and y is the target label.
Output: Optimized feature set $\mathcal{F}^*$.
Pre-Training Phase:
  Train the encoder-decoder model using a supervised loss function (cross-entropy or MSE).
  For each feature:
    Compute loss $L_{\text{classification}}$ or $L_{\text{regression}}$.
    Backpropagate and update model parameters.
  Store the best model with minimal validation loss as the initialization for PPO fine-tuning.
PPO Fine-Tuning Phase:
For each iteration $t = 1$ to $T_{\max}$:
  Sample transformation operations a from the policy $\pi_\theta(a|s)$.
  Apply transformations to $\mathcal{D}$ to generate a new feature set $\mathcal{D}^{\text{new}}$.
  Evaluate the performance of $\mathcal{D}^{\text{new}}$ on downstream task using a performance metric (e.g., accuracy, *F1-score*).
  Compute reward r based on the performance improvement:
$$r = \text{Metric}(\mathcal{D}^{\text{new}}) - \text{Metric}(\mathcal{D})$$
  Compute PPO objective:
$$L_{\text{PPO}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$
End for PPO Iterations

The PPO fine-tuning phase offers several advantages. Stability is achieved through PPO's clipping mechanism, which ensures stable training by limiting large updates to the policy. The reward-driven method allows the model to dynamically adapt to specific datasets and tasks.

4 Experiments

4.1 Experimental Setup

Data Description. We conducte experiments on 13 datasets from UCI [Public, 2024b], Kaggle [Howard, 2024], and OpenML [Public, 2024a], LibSVM [Lin, 2024], comprising 9 classification tasks and 4 regression tasks. Table 2 shows the statistics of the data.

Evaluation Metrics. For classification tasks, we use *F1-score*, accuracy, and precision to evaluate the dataset. For regression tasks, we use 1-relative absolute error (1-RAE) and R^2 to evaluate the dataset. The equation for 1-RAE is as fol-

Datasets	**ACC**						***F1-Score***						**Precision**					
classification	Base	GRFG	DFS	OpenFE	DIFER	TSFG	Base	GRFG	DFS	OpenFE	DIFER	TSFG	Base	GRFG	DFS	OpenFE	DIFER	TSFG
australian	0.8613	0.8759	0.8467	0.854	0.8467	**0.8832**	0.8613	0.8757	0.8466	0.854	0.8467	**0.8832**	0.8613	0.8765	0.8472	0.8544	0.8475	**0.8832**
credit_g	0.71	0.775	0.725	0.71	0.78	**0.785**	0.698	0.7756	0.7257	0.7116	0.7773	**0.7817**	0.6913	0.7763	0.7266	0.7133	0.7754	**0.7795**
diabetes	0.719	0.719	0.7059	0.732	0.7273	**0.7582**	0.7062	0.7115	0.6954	0.7233	0.7179	**0.7503**	0.7182	0.7151	0.7021	0.73	0.7182	**0.7583**
f5	0.795	0.76	0.7725	0.805	0.7925	**0.825**	0.7943	0.7602	0.7719	0.8034	0.7913	**0.824**	0.7956	0.7623	0.7726	**0.888**	0.7945	0.8273
hepatitis	0.8387	0.8387	0.871	0.8387	**0.9355**	0.871	0.8214	0.8215	0.8628	0.8342	**0.9355**	0.8762	0.8289	0.8289	0.8655	0.8318	**0.9355**	0.8895
ionosphere	0.9143	0.9	**0.9286**	0.9143	0.9143	**0.9286**	0.9125	0.8986	0.9266	0.9114	0.9126	**0.9267**	0.9176	0.9008	**0.9359**	0.9246	0.9176	**0.9359**
NPHA	0.4577	0.3803	0.3943	0.4225	0.4155	**0.4718**	0.4465	0.3914	0.3898	0.4199	0.4003	**0.4599**	0.4418	0.4091	0.3859	0.418	0.3896	**0.4534**
PimaIndian	0.719	0.7190	0.7059	0.732	0.7272	**0.7321**	0.7062	0.7115	0.6945	**0.7233**	0.7179	0.7111	0.7182	0.7151	0.7021	0.73	0.7182	**0.7461**
seismic	0.9167	0.9176	0.9205	0.9186	0.9201	**0.9205**	0.8854	0.8822	0.8936	0.8896	0.8896	**0.8961**	0.8793	0.8723	0.8965	0.889	**0.9076**	0.8968
	$1-\mathrm{RAE}$						R^2											
regression	Base	GRFG	DFS	OpenFE	DIFER	TSFG	Base	GRFG	DFS	OpenFE	DIFER	TSFG						
Openml_582	0.6551	0.5644	0.54	0.6117	0.6128	**0.6612**	0.849	0.7946	0.7735	0.8394	0.8147	**0.8556**						
Openml_595	0.5724	0.4327	0.5585	0.6067	**0.6193**	0.6162	0.7636	0.6825	0.8077	0.8385	**0.8444**	0.8114						
Openml_637	0.534	0.5407	0.5303	0.4631	0.5409	**0.5489**	0.7326	0.7708	**0.7709**	0.6718	0.7404	0.7444						
Openml_639	0	0.138	0	0	0	**0.1981**	0	0.1104	0	0	0	**0.1926**						

Table 1: Overall Performance. In this table, the best and second-best results are highlighted in bold and underlined fonts respectively. We evaluate classification tasks using *F1-score*, accuracy, and precision, and regression tasks using 1-RAE and R^2. The higher the value, the better the quality of the transformed feature set.

Datasets	**C/R**	**#Samples**	**#Features**
australian	C	690	14
credit_g	C	1000	21
diabetes	C	768	8
f5	C	267	44
hepatitis	C	155	19
ionosphere	C	351	34
NPHA	C	714	14
PimaIndian	C	768	8
seismic	C	210	8
Openml_582	R	500	25
Openml_595	R	1000	10
Openml_637	R	500	50
Openml_639	R	100	25

Table 2: Dataset Information.

lows:

$$1 - RAE = 1 - \frac{\sum_{i=1}^{n} |y_i - y_i^*|}{\sum_{i=1}^{n} |y_i - y_m|}, \tag{9}$$

y_i^* is the actual target value of the i-th observation, y_i is the predicted target value of the i-th observation, and y_m is the mean of all actual target values.

Baseline Methods. We compare our method with 5 widely used feature generation methods, as well as random generation and feature dimension reduction methods: (1) **Base**: using the original dataset without feature generation. (2) **GRFG** [Wang *et al.*, 2022]: iteratively generates new features and reconstructs an interpretable feature space through group-group interactions. (3) **DFS** [Kanter and Veeramachaneni, 2015]: an expansion-reduction method that first expands and then selects feature, automatically generated features for the dataset. (4) **DIFER** [Zhu *et al.*, 2022]: Performs automated feature engineering in a continuous vector space, introducing an encoder-predictor-decoder to optimize features. It optimizes embeddings along the gradient direction and recovers improved features from the optimized embeddings. (5) **OpenFE** [Zhang *et al.*, 2023]: Offers expert-level automated feature generation by integrating novel enhancement methods and a two-stage pruning algorithm, effectively identifying significant features.

Hyperparameter Settings. We utilize the Adam optimizer [Kingma and Ba, 2015] to optimize the PPO algorithm, with a learning rate set to 1×10^{-4}. The clipping parameter ϵ for PPO is set to 0.2, and an entropy coefficient of 1×10^{-4} was applied to promote exploration. The number of PPO iterations is 10. The split ratios for the training set, validation set, and test set are 0.6:0.2:0.2. The encoder-decoder model incorporated 8 attention heads, with an embedding vector dimension of 128 and a model hidden layer dimension of 128. The maximum sequence length for feature transformations was set to 100. During pre-training, a temperature scaling parameter T ranging from 0.1 to 1 is used to balance exploration and exploitation.

4.2 Overall Comparison

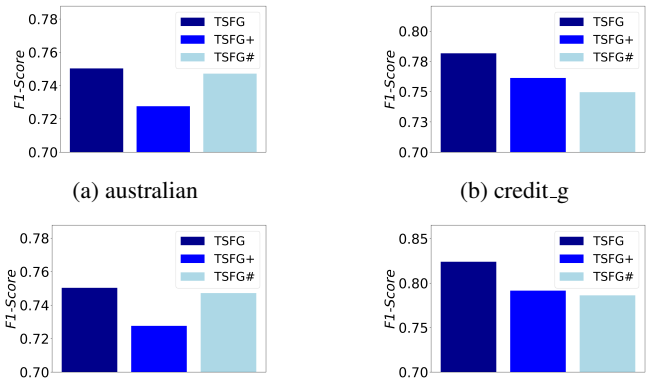
In this experiment, we compare the performance of TSFG and baseline models for feature transformation. We evaluate classification tasks using *F1-score*, accuracy, and precision, and regression tasks using 1-RAE and R^2. Table 1 shows the comparison results. We can see that in most cases, TSFG performs the best. By integrating the pre-training and fine-tuning stages, TSFG can accurately capture the internal patterns of the features, thereby identifying the optimal feature space. In conclusion, this experiment demonstrates the effectiveness of TSFG in feature transformation.

4.3 Ablation Study

To evaluate the contribution of each component in our framework, we perform ablation experiments. This experiment aims to verify whether each component of our method indeed has a positive impact on the final results. Therefore, we

have developed two variants: (1) No Pre-Training: The model is trained without the pre-training phase. This variant is referred to as "TSFG+". (2) No PPO Fine-Tuning: The model is evaluated without reinforcement learning fine-tuning. This variant is referred to as "TSFG#".

As shown in Figure 3, the results indicate that pre-training and PPO fine-tuning contribute significantly to the overall performance.



(a) australian (b) credit_g

(c) diabetes (d) f5

Figure 3: Results of ablation studies on different datasets.

4.4 Runtime Comparison

As shown in Table 3, we compare the runtime efficiency of the proposed framework with baseline methods. Despite incorporating reinforcement learning, our method achieves competitive runtime efficiency due to the use of pretraining. Compared with GRFG and DIFER, both of which use deep learning methods, the proposed method strikes a balance between computational cost and performance improvement.

Datasets	GRFG	DFS	OpenFE	DIFER	TSFG
australian	458	1	10	1310	92
credit_g	598	1	25	1335	114
diabetes	315	1	10	1950	104
f5	1655	5	55	3567	129
hepatitis	575	1	8	4000	114
ionosphere	1192	2	8	2217	130
NPHA	704	2	9	1573	53
Openml_582	1533	7	10	4615	70
Openml_637	2657	2	11	7136	102
Openml_639	553	2	19	2774	73
PimaIndian	306	2	9	2103	103
seismic	704	2	14	2160	139

Table 3: Time cost comparisons with baselines, in seconds.

4.5 Comparison on Different Downstream Tasks

To evaluate the adaptability and effectiveness of the proposed framework, we compare its performance across different downstream machine learning models, including Random Forest, XGBoost, Support Vector Machines (SVM), and CatBoost (CAT). The experiment is conducted on the diabetes dataset.

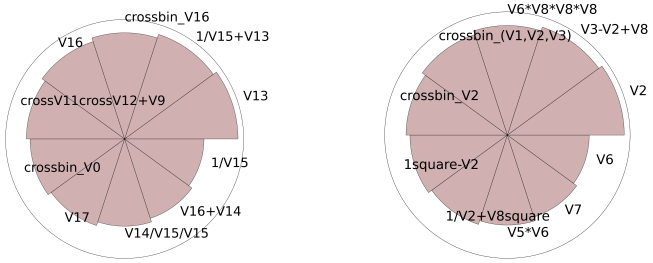
As shown in Table 4, the proposed framework performs well across different downstream models. This highlights the framework's ability to generate high-quality features whose effectiveness is not dependent on the complexity of the task or the choice of downstream model. Furthermore, it demonstrates the framework's adaptability to various modeling requirements and its potential for broad applications in the field of feature generation.

	RF	XGB	SVM	CAT
Base	0.7062	0.7276	0.6829	0.7469
GRFG	0.7115	0.7478	0.7059	0.7387
DFS	0.6945	0.755	0.6603	0.7335
OpenFE	0.7233	0.743	0.697	0.7115
DIFER	0.7179	0.7306	0.6893	0.7658
TSFG	0.7602	0.753	0.7275	0.7693

Table 4: Performance comparison of different downstream models.

4.6 Effectiveness of Generated Features

We evaluate the quality of the generated features by analyzing their importance in downstream models. Feature importance scores from tree-based models (e.g., Random Forest) are used to measure the contribution of generated features. Figure 4 shows that the generated features have high importance scores, often surpassing original features, indicating their effectiveness in capturing complex relationships in data.



(a) hepapits (b) credit_g

Figure 4: Feature importance analysis on different datasets.

5 Conclusion

In this paper, we propose a novel automated feature generation framework that integrates reinforcement learning with a Transformer-based encoder-decoder architecture. The framework addresses the challenges of feature redundancy, inefficient exploration of the feature space, and adaptability to diverse datasets and tasks. Our method dynamically generates high-quality features, enhancing the performance of downstream machine learning models. Extensive experiments on various datasets validate the effectiveness of the proposed framework. The framework provides a solid foundation for advancing automated feature generation.

Acknowledgements

This work was supported by the Science Foundation of Jilin Province of China under Grant YDZJ202501ZYTS286, and in part by Changchun Science and Technology Bureau Project under Grant 23YQ05.

References

[Bahdanau, 2014] Dzmitry Bahdanau. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[Cai *et al.*, 2023] Qingpeng Cai, Can Cui, Yiyuan Xiong, Wei Wang, Zhongle Xie, and Meihui Zhang. A survey on deep reinforcement learning for data processing and analytics. *IEEE Trans. Knowl. Data Eng.*, 35(5):4446–4465, 2023.

[Domingos, 2012] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

[Dong and Liu, 2018] Guozhu Dong and Huan Liu. *Feature engineering for machine learning and data analytics*. CRC press, 2018.

[Dor and Reich, 2012] Ofer Dor and Yoram Reich. Strengthening learning algorithms by feature discovery. *Information Sciences*, 189:176–190, 2012.

[Graves, 2013a] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[Graves, 2013b] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[Guo *et al.*, 2017] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR, 2017.

[He *et al.*, 2021] Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *Knowledge-based systems*, 212:106622, 2021.

[Horn *et al.*, 2020] Franziska Horn, Robert Pack, and Michael Rieger. The autofeat python library for automated feature engineering and selection. In *Machine Learning and Knowledge Discovery in Databases: International Workshops of ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part I*, pages 111–120. Springer, 2020.

[Howard, 2024] Jeremy Howard. Kaggle dataset download. https://www.kaggle.com/datasets, 2024. Accessed: 2024-05-01.

[Kanter and Veeramachaneni, 2015] James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE international conference on data science and advanced analytics (DSAA)*, pages 1–10. IEEE, 2015.

[Katz *et al.*, 2016] Gilad Katz, Eui Chul Richard Shin, and Dawn Song. Explorekit: Automatic feature generation and selection. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 979–984. IEEE, 2016.

[Khurana *et al.*, 2018] Udayan Khurana, Horst Samulowitz, and Deepak S. Turaga. Feature engineering for predictive modeling using reinforcement learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 3407–3414. AAAI Press, 2018.

[Kingma and Ba, 2015] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[Komer *et al.*, 2014] Brent Komer, James Bergstra, and Chris Eliasmith. Hyperopt-sklearn: Automatic hyperparameter configuration for scikit-learn. In *Scipy*, pages 32–37, 2014.

[LeCun *et al.*, 2015] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[Li *et al.*, 2024] Yonghao Li, Liang Hu, and Wanfu Gao. Multi-label feature selection with high-sparse personalized and low-redundancy shared common features. *Information Processing & Management*, 61(3):103633, 2024.

[Lin, 2024] Chih-Jen Lin. Libsvm dataset download. https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/, 2024. Accessed: 2024-05-01.

[Liu *et al.*, 2019] Kunpeng Liu, Yanjie Fu, Pengfei Wang, Le Wu, Rui Bo, and Xiaolin Li. Automating feature subspace exploration via multi-agent reinforcement learning. In Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis, editors, *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, pages 207–215. ACM, 2019.

[Luo *et al.*, 2019] Yuanfei Luo, Mengshuo Wang, Hao Zhou, Quanming Yao, Wei-Wei Tu, Yuqiang Chen, Wenyuan Dai, and Qiang Yang. Autocross: Automatic feature crossing for tabular data in real-world applications. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1936–1945, 2019.

[Obese, 1998] HJOR Obese. Body mass index (bmi). *Obes Res*, 6(2):51S–209S, 1998.

[Public, 2024a] Public. Openml dataset download. https://www.openml.org, 2024. Accessed: 2024-05-01.

[Public, 2024b] Public. UCI Dataset Download. https://archive.ics.uci.edu/, 2024. Accessed: 2024-05-01.

[Raffel *et al.*, 2020] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text trans-

former. *Journal of machine learning research*, 21(140):1–67, 2020.

[Schulman *et al.*, 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[Sutskever, 2014] I Sutskever. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*, 2014.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[Wang *et al.*, 2021] Qianqian Wang, Zhengming Ding, Zhiqiang Tao, Quanxue Gao, and Yun Fu. Generative partial multi-view clustering with adaptive fusion and cycle consistency. *IEEE Transactions on Image Processing*, 30:1771–1783, 2021.

[Wang *et al.*, 2022] Dongjie Wang, Yanjie Fu, Kunpeng Liu, Xiaolin Li, and Yan Solihin. Group-wise reinforcement feature generation for optimal and explainable representation space reconstruction. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1826–1834, 2022.

[Zhang and Gao, 2021] Ping Zhang and Wanfu Gao. Feature relevance term variation for multi-label feature selection. *Applied Intelligence*, 51:5095–5110, 2021.

[Zhang *et al.*, 2021] Ping Zhang, Wanfu Gao, Juncheng Hu, and Yonghao Li. A conditional-weight joint relevance metric for feature relevancy term. *Engineering Applications of Artificial Intelligence*, 106:104481, 2021.

[Zhang *et al.*, 2023] Tianping Zhang, Zheyu Aqa Zhang, Zhiyuan Fan, Haoyan Luo, Fengyuan Liu, Qian Liu, Wei Cao, and Li Jian. Openfe: automated feature generation with expert-level performance. In *International Conference on Machine Learning*, pages 41880–41901. PMLR, 2023.

[Zheng and Casari, 2018] Alice Zheng and Amanda Casari. *Feature engineering for machine learning: principles and techniques for data scientists*. " O'Reilly Media, Inc.", 2018.

[Zhu *et al.*, 2022] Guanghui Zhu, Zhuoer Xu, Chunfeng Yuan, and Yihua Huang. Difer: differentiable automated feature engineering. In *International Conference on Automated Machine Learning*, pages 17–1. PMLR, 2022.