Research on Machine Learning Program Generation Algorithm Based on AORBCO

Shiqian Wang
School of Computer Science and Engineering
Xi'an Technology University
Xi'an, China
E-mail:1178208937@qq.com

Songhan Wang
School of Computer Science and Engineering
Xi'an Technology University
Xi'an, China
E-mail:598226253@qq.com

Wuqi Gao
School of Computer Science and Engineering
Xi'an Technology University
Xi'an, China
E-mail:gaowuqi@126.com

Abstract*—The design and development of machine learning programs require selecting appropriate data and algorithms, and coding and debugging based on specific task requirements and the programming experience of developers. However, the current knowledge structure in the field of machine learning is relatively complex, lacking systematic organization, and developers often face the problem of lack of experience when choosing algorithms and designing programs, resulting in a long development cycle and easy errors in machine learning programs. In response to the above issues, this article proposes and designs a machine learning program generation algorithm based on the AORBCO model. The program generation ability includes two sub abilities: algorithm decision-making ability and code generation ability. AD-EKG has been designed for algorithmic decision-making ability, allowing Ego to select appropriate machine learning algorithms based on datasets in massive data. This algorithm combines the characteristics of the AORBCO model's domain knowledge base, knowledge graph based recommendation algorithm, and collaborative filtering algorithm. By calculating the descriptive and structural information between the dataset and algorithm, the interaction probability between the dataset and algorithm is obtained, allowing Ego to make algorithmic decisions interaction probability based. Results of the experiment have shown that the AD-EKG algorithm can fully utilize structural and descriptive information to improve the accuracy of Ego algorithm decision-making. CodeT5-EKG has been designed for code generation capability, allowing Ego to automatically generate machine learning program code. This algorithm combines the CodeT5 generative model with the domain knowledge base of the AORBCO model, by adding auxiliary information extracted using DPR technology to the code generation task, and performing diversified fusion operations to improve code generation quality. The CodeT5-EKG algorithm combines the creativity and efficiency of generative models and DPR technology, and is an algorithm that can improve the quality of generated code while also having the advantages of generative models. The experiments have proved that the code generated by this algorithm has better quality compared to other generative models with the same number of parameters.

Keywords-Program Generation; Recommendation Algorithms; AORBCO Model; Machine Learning

I. INTRODUCTION

The so-called automatic generation of machine learning programs is the process of providing a dataset by the user and letting the computer automatically generate machine learning algorithm code that meets the user's requirements based on the dataset. The challenges of time pressure and complexity in machine learning program development are addressed through automation and intelligence. However, due to the rapid growth of its associated data, machine learning technology, one of the most central techniques of artificial intelligence, is now facing a serious information overload problem [1].

DOI: 10.2478/ijanmc-2024-0013

For the past few years, researchers have attempted to use machine learning, deep learning, and other techniques to allow machines to automatically generate code. Beltramelli proposed a neural network model pix2code [2] which automatically reverse engineers the user interface and generates code based on GUI screenshots. Ahmad et al. proposed PLBART [3] pre-training model for program generation task. Wang et al. [4] raised a CodeT5 pre training model for code generation by incorporating tasks related to program identifiers during the pre-training process based on the principles of the T5 model [5]. OpenAI has released the ChatGPT model, which performs well in handling basic programming questions, answering technical questions, and generating basic code snippets. However, when dealing with complex and specific programming tasks, especially those that require a lot of details, ChatGPT's performance may have certain limitations.

In this context, the AORBCO model (Agent-Object-Relationship Model Based on Consciousness-Only) [6], as an intelligence model derived from the results of research on human intelligence consisting of the relevant theories of Consciousness-Only [7], and its idea of modeling knowledge can effectively alleviate the information overload problem. This model adopts the concept of "one person, one world" and proposes a reasonable abstraction of the objective world centered on Ego (self). From the perspective of human intelligence, thinking, and application, combining recommendation algorithms with code generation technology can leverage machine learning algorithms to improve user efficiency, thereby promoting the popularization and widespread application of machine learning algorithms.

Based on analyses and summaries of existing research, this paper proposes a machine learning program generation algorithm based on the AORBCO model. The program generation ability includes two sub abilities: algorithm decision-making ability and code generation ability. AD-EKG has been designed for algorithmic decision-making ability, allowing Ego to select appropriate machine learning algorithms based on datasets in massive amounts of data. Experimental results have shown that the AD-EKG algorithm can fully utilize structural and descriptive information to improve the accuracy of Ego algorithm decision-making. CodeT5-EKG has been designed for code generation capability, allowing Ego to automatically generate machine learning program code. The results of the experiment show that this algorithm generates higher quality code compared to other generative models with the same number of parameters.

II. Overall Design of Ego Program Generation Capability

The program generation capability of Ego refers to its ability to understand task requirements from user provided natural language descriptions and automatically complete the machine learning program generation process. The program generation ability includes two sub abilities: algorithm decision-making ability and code generation ability. The overall framework diagram of program generation capability is shown in Fig.1.

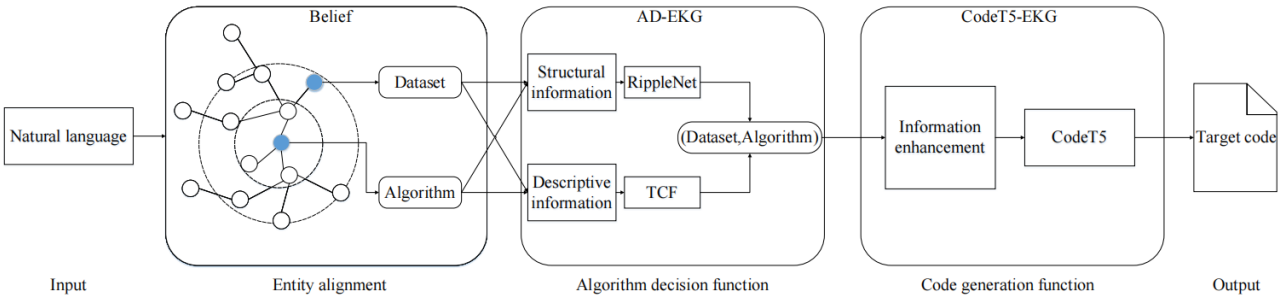


Figure 1. Overall framework diagram of program generation capability

Firstly, Ego receives natural language text sent by user Ego, which describes the characteristics of the dataset and the type of task. Subsequently, Ego utilized the knowledge alignment feature in the AORBCO model to align the knowledge of dataset objects in the domain knowledge base. At this stage, Ego will consider the dataset object described by user Ego to be the most similar object in the domain knowledge base after knowledge alignment. Then Ego will make capability calls for the object, including algorithm decision-making ability and code generation ability. The algorithmic decision-making ability will make decisions on the algorithmic object of the dataset, ultimately enabling Ego to find the algorithmic object that best matches the dataset object, forming a dataset algorithm binary. The subsequent code generation capability will be applied to the dataset algorithm binary for information augmentation and code generation, ultimately generating executable code. This end-to-end process enables Ego to understand task requirements from user provided descriptions and automatically complete the generation process of machine learning programs. Below, specific designs will be made for Ego's algorithm decision-making ability and code generation ability.

III. Design of AORBCO-ML Program Generation Algorithm

A. Design of Decision Ability for AD-EKG Algorithm

The algorithmic decision-making ability of Ego is its ability to select appropriate algorithms based on datasets in the knowledge graph of machine learning. This article designs an Algorithm Decision Based on Enhanced Knowledge Graph (AD-EKG) based on the characteristics of knowledge types in the domain knowledge graph of the AORBCO model. AD-EKG will combine structural information and descriptive information between objects to complete algorithm decision-making tasks, as shown in Fig.2.

AD-EKG includes a structural message calculation module and a descriptive information calculation module. The structural message calculation module is used to aggregate information from multi-level neighbors and extract structural information between objects. The descriptive information calculation module is used to extract linear and nonlinear relationships between object descriptive information. Below are two key components of AD-EKG.

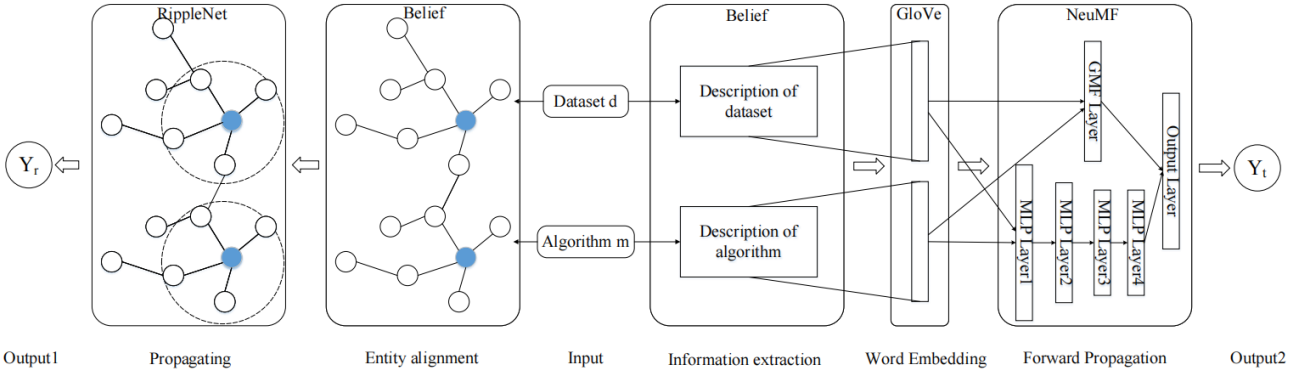


Figure 2. AD-EKG Overall Framework

1) Structural information calculation module

This article uses the RippleNet algorithm to implement the structural information calculation module of Ego. The input to the algorithm is a user item pair, the output is the probability of a user clicking on an item. This article views the dataset object as a user in a machine learning knowledge graph, an algorithm object as an item, and the relationship between the dataset object and the algorithm object as a historical interaction record. On this basis, use RippleNet to implement the algorithmic decision-making ability of Ego. The calculation process of RippleNet is shown in Fig.3.

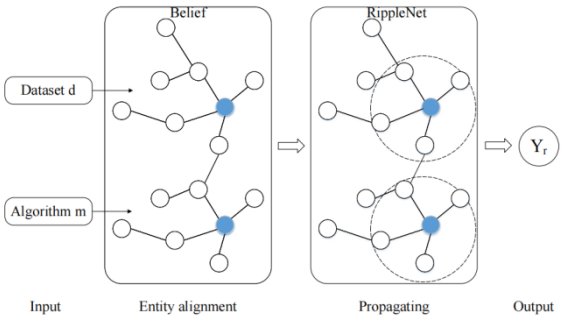


Figure 3. RippleNet calculation process

RippleNet calculates the interaction probability between two entities by searching for potential path information between user history click records and recommended items. The algorithm is executed as follows:

a) Model input: The model accepts users and items as inputs. User is represented by d, used to obtain the user's historical click history; The item is represented by m, representing the item to be predicted and clicked on.

b) Building a user seed set: The user seed set contains knowledge information that the user has operated on in the past. If it is a member of the seed set, the probability of clicking on the item during training is recorded as 1 (positive example), otherwise it is recorded as 0 (negative example).

c) First knowledge dissemination: obtain the set S_d^1 of first-order (hop-1) ripples of d, denoted by (h,r,t). To obtain valid recommendation information, there are certain invalid relationships between objects that need to be filtered.

d) Object embedding and similarity computation: normalized similarity is computed from the inner product of embedding vectors. The combination (h_i,r_i) of the head node h_i and the relation r_i in the first-level corrugated set S_d^1 given d is matrix-multiplied with the model input term m, and then the probability p_i of association of m with each (h_i,r_i) is obtained separately through the Softmax layer. Next, the model input term m is mapped into the embedding space, i.e., $m \in R^c$, and the dimension of the object embedding is denoted by c. The concrete representation of the association p_i probability is shown in equation(1).

$$p_i = \text{Softmax}(m^T R_i h_i) = \frac{\exp(m^T R_i h_i)}{\sum_{(h,r,t)\in S_d^1} \exp(m^T R_i h_i)} \quad (1)$$

At this point, the correlation probability p_i can be seen as the similarity between m and object h_i in the relationship space $R_i \in R^{c\times c}$, i.e., the degree to which the user's interests are preferred in the direction of the relationship in that r_i. Determine the correlation between m and each (h_i,r_i) based on the (h_i,r_i) in the ripple set S_d^k in the user's knowledge graph.

e) Calculate weighted average: In the previous step, the correlation probability p_i of m with respect to each (h_i,r_i) in the first level ripple set S_d^1 was obtained. p_i was multiplied by t_i in the first level ripple set S_d^1 and then summed to obtain the first order response o_d^1 of the current user d to m, as shown in equation(2).

$$o_d^1 = \sum_{(h,r,t)\in S_d^1} p_i t_i \quad (2)$$

Through the above process, the response of user hop-1 ripple set S_d^1 to m can be obtained. The process from step b to step e can be referred to as preference propagation.

f) Multiple knowledge propagation: The above steps are the first propagation of user's historical click records in the knowledge graph. To better mine knowledge, the first-order response o_d^1 obtained in step e is replaced by the embedded representation of m, and preference propagation continues. When the number of propagation is set to 3, the values of o_d^2 and o_d^3 can be calculated sequentially. Finally, the user's embedding representation is obtained by summing up the responses of each stage of d, as shown in equation(3).

$$d = o_d^1 + o_d^2 + ... + o_d^H \quad (3)$$

g) Predictive value calculation: The user embed representation and the item embed representation are inner products, and the predicted value Y_r is obtained through the Sigmoid function, which means the probability of user d clicking on item m, as shown in equation(4) Where σ represents the Sigmoid activation function.

$$Y_r = \sigma(d^T m) \quad (4)$$

2) Descriptive information calculation module

For the implementation of the descriptive information computing module, this paper proposes the TCF (Text-based Collaborative Filtering) algorithm that makes improvements to the NCF (Neural Collaborative Filtering) algorithm [8], which uses the NeuMF to jointly train the GMF and MLP, and utilizes text data to achieve recommendations. For joint training and utilizes text data to achieve recommendations. The algorithm takes descriptive information about the user-item as input and the output is the probability of the user clicking on the item.

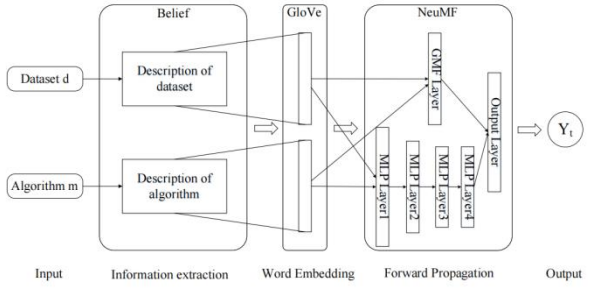


Figure 4. TCF Calculation Process

The overall structure of TCF is shown in Fig.4 above. Specifically, using t to denote the initial input text and n to denote that there are n words in the initial input text, then the descriptive text corresponding to the object can be denoted by $t = w_{1:n} = [w_1, w_2, ..., w_n]$. In this paper, we use GloVe [9] to initialize the embedding representation of each word w_i and obtain the sentence representation s by accumulating the representations of each word.

After text embedding, this paper obtains the vector forms s_d and s_m of the dataset and the descriptive text of the algorithm. In order to dig deeper into the interaction information between the two feature vectors, this paper uses the GMF and MLP layers to analyze the linear and nonlinear correlations of the dataset and the algorithm descriptive features, and then fuses these two types of information using the NeuMF layer to calculate the interaction probability of the dataset and the algorithm.

The linear interaction between the dataset and the algorithm description features can be obtained through the GMF layer as shown in equation (5) and equation (6) below:

$$\varphi_1 = s_d \odot s_m \quad (5)$$

$$y_1 = \sigma(G^T \varphi_1) \quad (6)$$

Here, $\odot$ denotes the product of elements. G^T is the weight matrix that can be obtained through learning. σ Denotes the Sigmoid activation function. This step can be interpreted as a special kind of matrix decomposition and has a higher expressive power than its original form.

While obtaining the linear interaction describing the features, this paper obtains the nonlinear interaction relationship between the two through the MLP layer. Specifically, this article connects two feature vectors and captures nonlinear interactions between features through a multi-layer fully connected network. The equation (7, 8 and 9) are as follows:

$$h_0 = s_d \,||\, s_m \quad (7)$$

$$\varphi_{nl} = h_n = \sigma(W_n^T h_{n\text{-}1} + b_n) \quad (8)$$

$$y_{nl} = \sigma(W^T \varphi_{nl}) \quad (9)$$

The symbol || represents the concatenation operation between feature vectors. In order to integrate linear and nonlinear interactions of text

features, this paper connects the last layer of GMF and MLP, and fuses linear φ_l and nonlinear feature φ_{nl} through NeuMF layer to better learn implicit interactions between descriptive texts and predict the final interaction probability of d and m. Its equation (10) is as follows:

$$Y_t = \sigma(W_t^T(\varphi_l \| \varphi_{nl})) \quad (10)$$

Finally, combining the two probabilities, equation (11) is as follows:

$$Y = W^T(\sigma(d^T m) + \sigma(W_t^T(\varphi_l \| \varphi_{nl}))) \quad (11)$$

Among them, $\sigma(d^T m)$ calculates the interaction probability between objects based on their structural information, $\sigma(W_t^T(\varphi_l \| \varphi_{nl}))$ calculates the interaction probability between objects based on their descriptive information, and W^T is the weight matrix.

B. Design of CodeT5-EKG code generation capability

The code generation capability of Ego is the ability to generate corresponding code based on datasets and algorithms. In order to build the code generation capability of Ego, this article uses the CodeT5+model as the basic model and integrates the domain knowledge base of the AORBCO model as auxiliary information for the generative model during code generation. A knowledge enhanced code generation algorithm (CodeT5-EKG) is constructed, which can improve the quality of generated code and has the advantages of generative modelling, as shown in Fig.5.

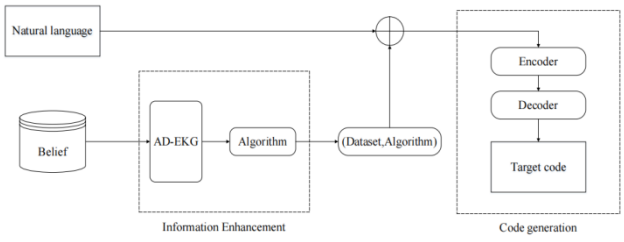


Figure 5. A Code Generation Algorithm Framework Based on Knowledge Enhancement

CodeT5-EKG consists of a code generation module and an information augmentation module. The code generation module is used to convert machine learning code templates into corresponding machine learning program code. The information enhancement module is used to extract relevant code from the domain knowledge base as auxiliary information for the code generation module, thereby improving the performance of the code generation module. Below are two key components of CodeT5-EKG.

1) Code generation module

In this paper, the CodeT5 family of models is used as the basic model for code generation, on the basis of which further innovations are made to obtain the CodeT5+ model. First, the model introduces a flexible mode selection mechanism, which enables it to run flexibly in encoder-only, decoder-only, or encoder-decoder modes according to the needs of different tasks. This design makes CodeT5+ more adaptable to different types of downstream tasks and improves the generality of the model. Second, CodeT5+ employs a multi-task pre-training strategy, including diverse tasks such as span denoising, causal language modeling (CLM), and text-code comparison learning. Such a set of pre-training tasks helps the model learn richer representations from both code and text data, allowing for better migration and adaptation in various applications.

In terms of model architecture, CodeT5+ adopts a "shallow encoder and deep decoder" architecture. The encoder and decoder get initialized by pre-training checkpoints and connected to the cross-attention layer. By freezing the deep decoder and training only the shallow encoder and the cross-attention layer, the computational efficiency is improved while the performance of the model is maintained. In addition, CodeT5+ introduces mechanisms for adjusting instructions to better align with natural language instructions. This mechanism makes the model more flexible in understanding and following natural language instructions, thus better meeting user expectations when generating code.

The CodeT5+ model was trained using the expanded CodeSearchNet pre-training dataset, which contains nine programming languages, as shown in Table 1.The model was divided into two groups for pre-training, the first group being CodeT5P-220M, CodeT5P-770M, and the second group is CodeT5P-2B, CodeT5P-6B, and CodeT5P-16B.The first group is trained from scratch according to the architecture of T5; while in the second group, the decoders of the models are initialized from the CodeGen-mono-2B, CodeGen-mono-6B, CodeGen-mono- 16B models were initialized, and the encoder was initialized from the CodeGen-mono-350M model.

TABLE I. PRE-TRAINING DATASET

Language	Sample quantity
Ruby	2,119,741
JavaScript	5,856,984
Go	1,501,673
Python	3,418,376
Java	10,851,759
PHP	4,386,876
C	4,187,467
C++	2,951,945
C#	4,119,796

In terms of model pre-training, CodeT5+ adopts two stages for pre-training: in the first stage of pre training, the model undergoes pre-training for span denoising tasks and joint training for two CLM tasks, and uses a linear decay learning rate (LR) scheduler with a maximum learning rate of 2e-4. The batch size of the denoising task is set to 2048, while the batch size of the CLM task is 512. In the second stage of pre-training, the model adopted a strategy of equal weight contrastive learning, matching, and joint optimization of two CLM losses, and underwent 10 cycles of training. Set the batch size to 256 and the learning rate to 1e-4. The maximum length of the code and text sequence is set to 420 and 128, respectively. The model uses the AdamW optimizer weights decay to 0.1. At the same time, the mixed precision training technique of ZeRO Stage 2 and FP16 using DeepSpeed [10] is utilized to accelerate the training process.

2) Information Enhancement Module

When facing problems, Ego usually consults and organizes relevant information in the knowledge base to enhance the specificity and accuracy of the answers. In recent years, some researchers have attempted to incorporate knowledge bases into generative tasks and perform diverse fusion operations to improve the efficiency of algorithms. They proposed a hybrid neural dialogue model with both response retrieval and generation capabilities. Lewis et al. [11] proposed a RAG framework for knowledge intensive NLP tasks, which utilizes the DPR(Dense Passage Retrieval) algorithm to extract information from search results, concatenates the extracted information with the original input, and finally inputs the concatenated results into a generator for processing [12]. Experimental results have shown that this method can produce more specific and accurate results.

In order to fully utilize DPR technology and its advantages in natural language processing and information retrieval, this paper adopts DPR technology to achieve code extraction of Ego in the domain knowledge base. DPR uses a text encoder to encode the questions and answers in question and answer data separately to convert the input text into a dense vector representation. By calculating the similarity between the two vectors to evaluate their correlation, it achieves fast retrieval in large-scale text datasets.

This paper constructs an information enhancer specifically designed for machine learning code generation tasks based on DPR technology. In DPR, by using question answer pairs as training data, the model can learn how to accurately match the correlation between questions and answers, thereby improving the accuracy of retrieval. Two of the encoders used pre trained CodeBERT to obtain better vector representations. Its structure is shown in Fig.6.

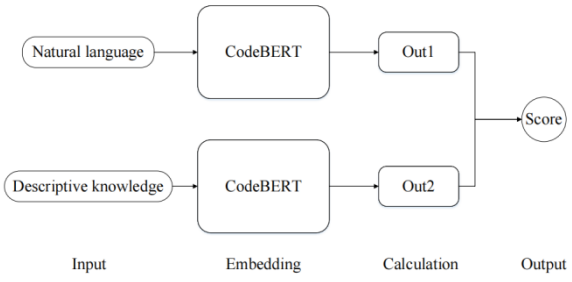


Figure 6. Diagram of DPR-based enhancer architecture

The objective likelihood function of the enhancer can be expressed as equation (12):

$$L(q_i, c_i^+, c_{i,l}^-, ..., c_{i,n-l}^-) = -\log \frac{e^{\text{sim}(q_i, c_i^+)}}{e^{\text{sim}(q_i, c_i^+)} + \sum_{j=1}^{n-1} e^{\text{sim}(q_i, c_{i,j}^-)}} \quad (12)$$

Among them, q_i represents the i th natural language input, c_i^+ refers to the correct descriptive information fragment related to natural language q_i, $c_{i,j}^-$ represents the j th descriptive information block except for c_i^+, n means the total number of samples, and sim stands for the calculation of dot product similarity. After being processed by an information enhancer, the processing method of Izacard et al [14] is referenced to splice and replace natural language inputs with descriptive chunks of information The process is shown in equation (13):

$$x' = x \oplus y_1 \oplus y_2 \oplus ... \oplus y_n \quad (13)$$

Where x denotes the original input text, y_k denotes the k th spliced and replaced descriptive information block, $\oplus$ denotes the splicing and replacing operation, and x' denotes the spliced and processed input text. The original input text for the CodeT5+ model is shown in Fig.7 below.

```
code_framework = f"""Write code for {task_area} to load and partition the {dataset_name} dataset,
define, train and evaluat {algorithm_name} models:
# Import necessary libraries
[EKG1]

# Load dataset
[EKG2]

# Split dataset
[EKG3]

# Initialize model
[EKG4]

# Train model
[EKG5]

# Evaluate model
[EKG6]

# Use the model for predictions on new data
[EKG7]
"""
```

Figure 7. original input

The above figure shows the original input text of the CodeT5+ model. Among them, task area represents the domain of the machine learning problem, dataset name represents dataset object's name, and algorithm name represents the algorithm object's name. In addition, the original input also includes module annotations related to machine learning programs, such as importing third-party libraries, loading and splitting datasets, model definitions, etc. The annotation texts of each module are connected with placeholders "[EKG]".

When performing information augmentation, the relevant fields such as domain, dataset name, algorithm name, and placeholder "[EKG]" will be replaced by the algorithm selected by the Ego algorithm's decision-making ability and the relevant information retrieved by DPR, forming the input source data after replacement processing. Partial retrieval information examples and text replacement examples are shown in Fig.8 and Fig. 9.

```
<attr name="datasetImportLib" weight="0.1">
    <![CDATA[
import os
import numpy as np
from tensorflow.keras import layers
from tensorflow.keras import models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
    ]]>
</attr>
```

Figure 8. Retrieving information Example

```
Write code for image classification to load and partition the Intel Image Classification dataset,
define, train and evaluat ResNet models:
# Import necessary libraries
import os
import numpy as np
from tensorflow.keras import layers
from tensorflow.keras import models
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load dataset
[EKG2]

# Split dataset
[EKG3]

# Initialize model
[EKG4]

# Train model
[EKG5]

# Evaluate model
[EKG6]

# Use the model for predictions on new data
[EKG7]
```

Figure 9. Text Replacement Example

When DPR fails to retrieve the corresponding text, the CodeT5+ model will directly generate code and replace the corresponding part of the placeholder "[EKG]". After DPR retrieval replacement and CodeT5+ model generation replacement, the original input will become a

complete code sequence. The final example of code generation is shown in Fig.10.

The above methods combine the advantages of information retrieval and generative models. DPR can quickly and accurately retrieve relevant code fragments, providing rich contextual information and prior knowledge. The retrieved code snippets help CodeT5+better grasp the context and generate code that matches the task requirements.

```
# Import necessary libraries
import os
import numpy as np
from tensorflow.keras import layers
from tensorflow.keras import models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf
from tensorflow.keras.applications import ResNet50

# Load dataset
# Set up data directories
train_dir = '/path/to/train'
validation_dir = '/path/to/validation'
test_dir = '/path/to/test'

# Set up data generators
train_datagen = ImageDataGenerator(rescale=1./255)
validation_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical')
```

Figure 10. Code Generation Example

IV. EXPERIMENTAL RESULTS AND ANALYSIS

A. Verification of Decision Ability of AD-EKG Algorithm

This section mainly conducts validation experiments on the decision-making ability of the Ego algorithm, and the environmental information studied in the experiments is shown in Table 2.

TABLE II. EXPERIMENTAL ENVIRONMENT INFORMATION

Name	Configuration information
operating system	Windows 11
RAM	16G
Graphics card	NVIDIA GeForce RTX 3070 8G
development language	Python 3.7.8
Deep learning platform	TensorFlow 2.2.0

After studying the characteristics of data in the field of machine learning and the classification strategies of machine learning related information resources and network platforms, this article uses web scraping technology to collect data from websites such as Paperswithcode and Github. These data mainly include datasets, algorithms, and other related objects related to the field of machine learning, forming a knowledge graph based recommendation algorithm dataset. The dataset constructed in this article covers four fields (computer vision, semantic segmentation, image generation, and object detection). Including 256 machine learning datasets, 1482 machine learning algorithms, 4 machine learning tasks, 1366 academic papers, etc., a total of 5314 objects.

1) Building a dataset

After cleaning and preprocessing the crawled data, this article successfully screened 233 machine learning datasets and 1448 machine learning algorithms, which will be used for training models and analyzing user item interactions. As shown in Table 3.

TABLE III. DATASET STATISTICS

Domain knowledge graph		Dataset	
Number of objects	5262	Number of dataset objects	233
Relationship types	48	Number of algorithm objects	1448
Number of triples	14774	Number of interactions	1485
Average number of descriptive words	50.5	Sparsity	0.00440

2) Experimental plan

a) Evaluation indicators. This article models the decision-making ability of Ego algorithm as a recommendation algorithm, and in recommendation algorithms, the recommended results are usually viewed as a classification problem, that is, whether users like the items recommended by the recommendation system. Therefore, this article adopts commonly used indicators, including AUC, Precision, Recall, F1 score, and NDCG.

b) Parameter settings.

For RippleNet, The object embedding and relation embedding dimensions are configured to 16, with a maximum of 3 hops, 10 epochs, and a batch size of 32, optimized using the Adam optimizer. The learning rate and regularization coefficients are determined via grid search, and the search spaces are {10-4, 5×10-4, 10-3, 5×10-3}和{10-5, 10-4, 10-3, 10-2}；

For TCF, the dimension of the text embedding was set to 300, Multiply was used in GMF for linear computation, and 4 fully connected layers were used in MLP for nonlinear computation, and the outputs of GMF and MLP were connected by Concatenate of NeuMF.

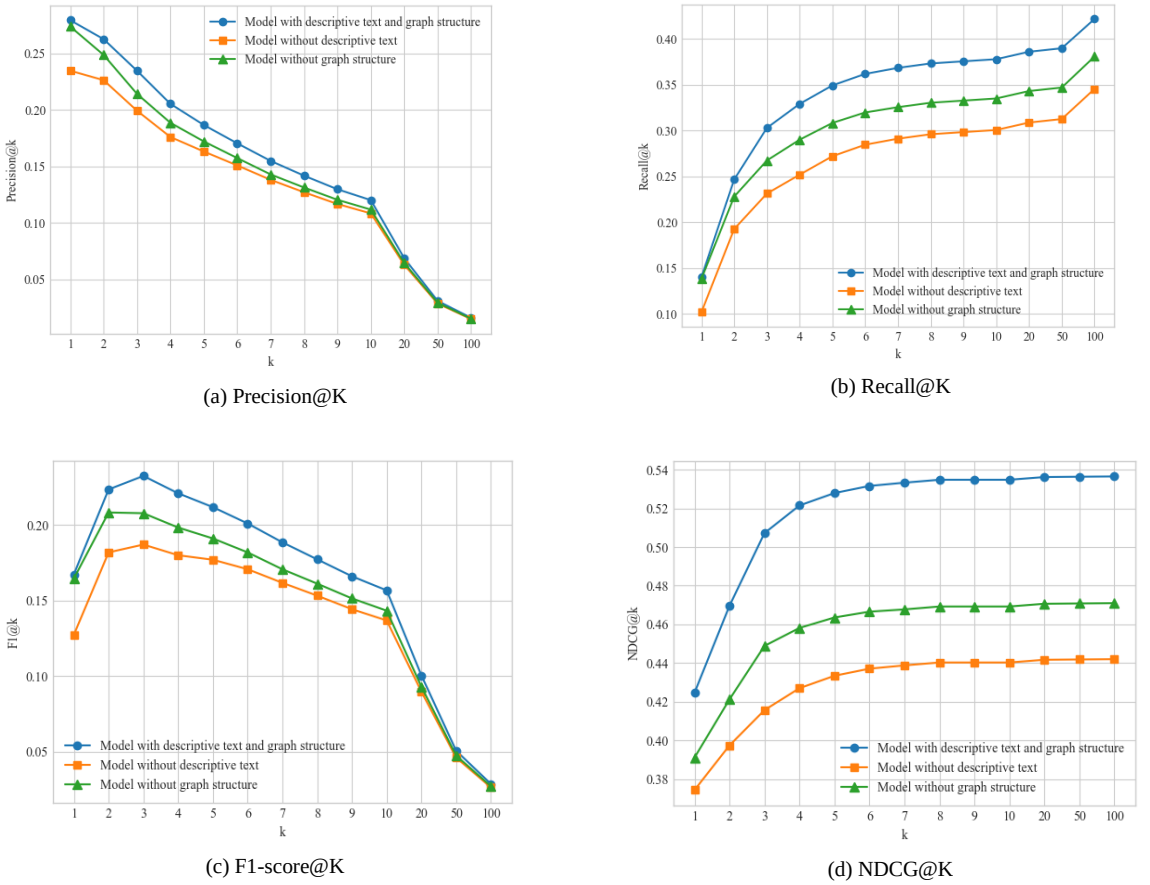
c) Comparison experiment. We compare AD-EKG with KGNN-L[14] and KGCN [15]recommendation models

d) Ablation experiment. To investigate the validity of the algorithm, i.e., whether both graph structural information and textual descriptive information are helpful for recommendation, this paper sets up the following scenarios for Top-K evaluation:

- Using only structural information (RippleNet);
- Using only descriptive information (TCF);
- Using both structural and descriptive information (AD-EKG).

3) Experimental analysis

The results of AD-EKG on the CTR prediction task and Top-K's recommendation are shown in Fig.11 and Table 4, respectively.



(a) Precision@K

(b) Recall@K

(c) F1-score@K

(d) NDCG@K

Figure 11. Top-K ablation experiments of AD-EKG under different variants

TABLE IV. CTR PREDICTION COMPARISON EXPERIMENT (%)

Model	AUC	Precision	Recall	F1-score
KGNN-LS	80.01	71.63	76.10	73.80
KGCN	71.62	62.78	64.38	63.57
RippleNet	82.55	69.43	86.91	77.19
TCF	82.16	78.24	82.81	80.46
AD-EKG	88.20	83.80	86.82	85.28

The experimental results from the experiments are presented in Table 4 and Fig.11. From the metrics Precision, Recall, F1-score and NDCG, AD-EKG outperforms the model that does not use both information in the recommendation task.The Recall value of AD-EKG increases with the length of the recommendation list, which indicates that the model is better able to capture the user's interests and needs.The NDCG metric is a ranking quality and relevance to measure the performance of ranking models in recommendation algorithms, AD-EKG is also higher than traditional models in NDCG metrics, indicating that AD-EKG can provide more relevant and higher quality recommendation results.

In summary, the AD-EKG model outperforms the single method traditional model on the CTR prediction task and Top-K recommendation. This suggests that the simultaneous use of structural and descriptive message from the knowledge graph can significantly improve the effectiveness of recommendation models.

B. Validation of CodeT5-EKG Code Generation Capabilities

This section focuses on the validation experiments of Ego code generation capability. Considering the performance requirements of the large language model, the experiments in this section are chosen to be conducted on the cloud platform. The specific environment information of the cloud platform is shown in Table 5 below.

TABLE V. CLOUD PLATFORM EXPERIMENTAL ENVIRONMENT INFORMATION

Name	Configuration information
operating system	Ubuntu 20.04.5 LTS
memory	64G
graphics card	NVIDIA A100 40GB
development language	Python 3.8
Deep learning platform	Pytorch 2.0.0

1) Dataset

In order to verify the performance of the DPR technique on the code generation task, this paper constructs a dataset of questions related to machine learning program generation.

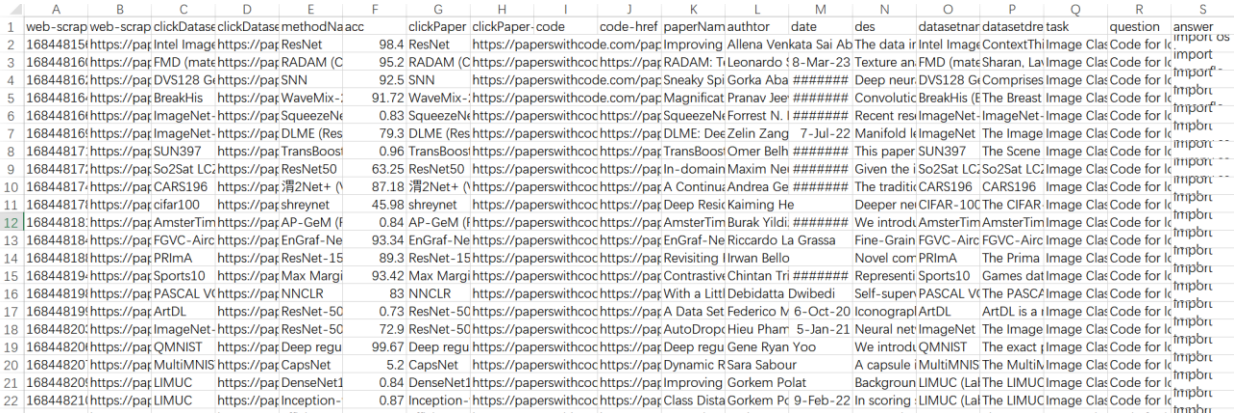


Figure 12. Example plot of a sample dataset

The dataset mainly consists of 122 question and answer data on machine learning image classification questions, as shown in Fig.12 below. The dataset of the machine learning program constructed in this paper to generate relevant questions is shown in Fig.12 above, where columns 3, 5, 14, 16, 17, and 18 of the file correspond to the dataset, algorithm, description

of the algorithm, description of the dataset, type of the task, relevant questions, and the answers of the machine learning domain, respectively, and the detailed data about the question and answer section is shown in Table 6.

TABLE VI. STATISTICAL DATA ON Q&A DATASET

Dataset	Attribute
source language	English
target language	Python
quantity	121
Average number of words in the source language	52
Maximum number of words in the source language	69
Average number of words in the target language	1365
Maximum number of words in the target language	1593

2) Evaluation Metrics

In this paper, the CodeBLEU metric [16] and ROUGE [17] metrics are used for assessing the quality of the code generated by the model. The CodeBLEU metric is a variant of the BLEU (Bilingual Evaluation Understudy) metric [18], and the BLEU metric is calculated as follows:

$$\mathrm{BLEU} = \mathrm{BP} \cdot \exp(\sum_{n=1}^{N} w_n \log P_n) \tag{14}$$

$$\mathrm{BP} = \begin{cases} 1 & if \quad c > r \\ e^{1-r/c} & if \quad c \le r \end{cases} \tag{15}$$

w_n denotes the weight of the n-tuple and p_n is the precision of the co-occurring n-tuple.BP is a penalty factor used to ensure that the scoring takes into account the length of the generated sequence and does not just focus on how accurate the generation is.

CodeBLEU is based on BLEU, additional syntactic matching as well as semantic matching score items are introduced, and the final score is weighed by a certain proportion, and its calculation formula16 is as follows:

$$\mathrm{CodeBLEU} = \alpha \cdot \mathrm{BLEU} + \beta \cdot \mathrm{BLEU}_{\mathrm{weight}} + \gamma \cdot \mathrm{Match}_{\mathrm{ast}} + \delta \cdot \mathrm{Match}_{\mathrm{df}} \tag{16}$$

In BLEU calculation, different tokens have the same weight, and different tokens have different weights in CodeBLEU calculation. In equation (16), $\mathrm{BLEU}_{\mathrm{weight}}$ is a weighted n-gram matching metric, similar to the BLEU computation; $\mathrm{Match}_{\mathrm{ast}}$ is the similarity of the abstract syntax tree, which is used to measure the syntactic information of the code; and $\mathrm{Match}_{\mathrm{df}}$ is the similarity of the semantic data flow, which takes into account the semantic similarity between the generated code and the reference code.

ROUGE metrics are mainly used to measure the degree of overlap between computer-generated code and reference code to evaluate assess the quality of automatically generated code. Commonly used evaluation metrics include ROUGE-N and ROUGE-L.

ROUGE-N mainly evaluates the code quality by calculating the number of n-grams that are the same in all the sentences in the automatically generated code and the reference code, and the proportion of them in the reference code. The detailed calculation formula17 is given below:

$$\mathrm{ROUGE\text{-}N} = \frac{\sum_{S \in \{Reference\}} \sum_{gram_N \in S} Count_{match}(gram_N)}{\sum_{S \in \{Reference\}} \sum_{gram_N \in S} Count(gram_N)} \tag{17}$$

$gram_N$ means that the length of the word is n, $Count_{match}(gram_N)$ represents the frequency with which words of length n exist both within the automatically generated code and within the reference code, as opposed to $Count(gram_N)$ which represents the frequency with which words of length n exist only within the reference code.

ROUGE-L counts the longest common substring that exists between the automatically generated code and the reference code to evaluate the overall coherence of the code, with Eqs. (18, 19, and 20) as follows:

$$R_{lcs} = \frac{LCS(X,Y)}{m} \tag{18}$$

$$P_{lcs} = \frac{LCS(X,Y)}{n} \tag{19}$$

$$F_{lcs} = \frac{(1+\beta^2)LCS(X,Y)}{R_{lcs} + \beta^2 P_{lcs}} \qquad (20)$$

Equation (19) and equation (20) denote the calculation of recall R_{lcs} and accuracy P_{lcs}, respectively. The F_{lcs} in equation (21) denotes the final calculated ROUGE-L value. Where X denotes the text of the reference code, and its length is identified by m. Y denotes the text of the model-generated code, and its length is identified by n. $LCS(X,Y)$ denotes the length of the longest common subsequence of X and Y. The parameter β is generally set to a larger number, which is used to indicate that the calculation of P_{lcs} recall holds a larger weight in the calculation of F_{lcs}.

3) Experimental analysis

Comparison of the experimental results is shown in Table7. It indicates that combining DPR technology with generative models is more effective in handling code generation problems than pure generative models when using the same parameter quantity model.

TABLE VII. COMPARATIVE EXPERIMENT (%)

label	model	Parameter quantity	CodeBLEU	ROUGE-1	ROUGE-2	ROUGE-L
1	CodeT5	770M	12.62	7.62	3.02	5.29
2	CodeT5-EKG	770M	23.93	13.52	4.62	10.02
3	CodeT5	2B	32.83	20.04	6.43	14.32
4	CodeT5-EKG	2B	47.94	24.30	9.22	17.60
5	CodeT5	6B	46.27	32.96	14.21	25.68
6	CodeT5-EKG	6B	51.12	35.58	16.11	27.54

TABLE VIII. COMPARISON WITH OTHER MODELS (%)

label	model	Parameter quantity	CodeBLEU	ROUGE-1	ROUGE-2	ROUGE-L
1	CodeT5-EKG	770M	23.93	13.52	4.62	10.02
2	CodeT5-EKG	2B	47.94	24.30	9.22	17.60
3	CodeT5-EKG	6B	51.12	35.58	16.11	27.54
4	CodeGen-Mono	2B	34.08	20.23	6.52	14.94
5	GPT-Neo	2.7B	19.82	12.57	2.79	11.28
6	InstructCodeT5	16B	43.71	25.00	9.63	21.06

Analyze the results in Table 8. As the number of model parameters increases, CodeT5-EKG shows significant improvements in both CodeBLEU and ROUGE metrics. Compared to purely generative models such as CodeGen Mono and GPT Neo, CodeT5-EKG exhibits higher code generation accuracy and consistency at smaller parameter sizes. In conclusion, the comparative results in Table 8 show that combining DPR techniques with generative models has significant advantages in English code tasks.

The retrieval + generative model constructed in this article combines the efficiency of the retrieval model with the creativity of the generative model. This combination enables the model to better control the generated content and make the generated content more reasonable. Compared to pure generative models, retrieval + generative models require fewer parameters and computational resources, making them easier to train and deploy. However, this model also has some limitations. It relies on information from prior data for retrieval, so different prior knowledge needs to be stored for different fields or tasks. Secondly, this behavior of retrieval may lead to a lack of diversity in the generated content, which may not be as flexible as pure generative models in some cases.

V. CONCLUSIONS

This article details the design and validation of a programme generation method based on the AORBCO model, including the design of algorithm decision-making ability and code generation ability. In the design of algorithm decision-making ability, this article proposes the AD-EKG algorithm. This algorithm combines the

characteristics of AORBCO model's domain knowledge graph, RippleNet, and TCF algorithm to enable Ego to intelligently select machine learning algorithms suitable for different tasks and datasets. The experimental results show that the AD-EKG algorithm can intelligently select suitable machine learning algorithms on different tasks and datasets, providing reliable decision-making basis for automatic program generation. In the design of code generation capability, this article adopts CodeT5+as the basic model for program generation. CodeT5+ is a pre-trained converter architecture that combines the information enhancer DPR to transform abstract algorithm descriptions into executable code. The experimental results show that the code generated by the CodeT5-EKG model has good accuracy and readability, providing support for the practicality of automatic generation of machine learning programs.

This article proposes a novel machine learning program automatic generation algorithm in the context of the AORBCO model, which has made important contributions to promoting research and application in the field of automated machine learning program design. In future research, the method proposed in this article can be further optimized and expanded to better adapt to the needs of different fields and tasks, providing more possibilities for the development of artificial intelligence.

REFERENCES

[1] Huang Liwei, Jiang Bitao, Lu Shouye et al. Review of recommendation systems based on Deep Learning [J]. Journal of Computers, 2018, 41(07):1619-1647.

[2] Beltramelli T. pix2code: Generating Code from a Graphical User Interface Screenshot [C]//Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems. 2018: 1-6.

[3] Ahmad W U, Chakraborty S, Ray B, et al. Unified Pre-training for Program Understanding and Generation [J]. 2021. DOI: 10.18653/v1/2021.naacl-main.211.

[4] Wang Y, Wang W, Joty S, et al. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation [J]. 2021.

[5] Raffel C, Shazeer N, Roberts A, et al. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer [J]. 2019. DOI:10.48550/arXiv.1910.10683.

[6] Feng Yanxing, Research on Program Generation in AORBCO Model [D]. Xi'an University of Technology, 2021. DOI:10.27391/dcnki.gxagu.2021.000121

[7] Xiao Liangshun, Research on Knowledge Fusion in AORBCO Modeling [D]. Xi'an University of Technology, 2023.

[8] He X, Liao L, Zhang H, et al. Neural Collaborative Filtering [J]. International World Wide Web Conferences Steering Committee, 2017. DOI:10.1145/3038912.3052569.

[9] Pennington J, Socher R, Manning C. Glove: Global Vectors for Word Representation [J]. 2014. DOI:10.3115/v1/D14-1162.

[10] Rasley J, Rajbhandari S, Ruwase O, et al. DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters [C]//KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. ACM, 2020. DOI:10.1145/3394486.3406703.

[11] Lewis P, Perez E, Piktus A, et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks [J]. 2020. DOI:10.48550/arXiv.2005.11401.

[12] Karpukhin V, Ouz B, Min S, et al. Dense Passage Retrieval for Open-Domain Question Answering [J]. 2020. DOI:10.18653/v1/2020.emnlp-main.550

[13] Izacard G, Grave E. Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering [J]. 2020. DOI:10.48550/arXiv.2007.01282.

[14] Wang H, Zhang F, Zhang M, et al. Knowledge-aware Graph Neural Networks with Label Smoothness Regularization for Recommender Systems [J]. SIGKDD explorations, 2019.

[15] Li Xiang, Yang Xingyao, Yu Jiong et al. A bipartite recommendation algorithm based on knowledge graph convolutional networks [J]. Computer Science and Exploration, 2022, 16(01):176-184.

[16] Ren S, Guo D, Lu S, et al. CodeBLEU: a Method for Automatic Evaluation of Code Synthesis [J]. 2020. DOI:10.48550/arXiv.2009.10297.

[17] Barbella, Marcello and Tortora, Genoveffa, Rouge Metric Evaluation for Text Summarization Techniques. Available at SSRN: https://ssrn.com/abstract=4120317

[18] Ehud Reiter; A Structured Review of the Validity of BLEU.Computational Linguistics 2018; 44 (3): 393–401. doi:https://doi.org/10.1162/coli_a_00322