2025 International Conference on Intelligent Computing
July 26-29, Ningbo, China
https://www.ic-icc.cn/2025/index.php

Cross-Domain Functional Knowledge Integration Architecture Powered by Deep Reinforcement Learning

Ding Yishen[1], Hu Yahong[1,*], Xie Youbai[2,3] , Meng Xianghui[2], Mao Jiafa[1]

[1]College of Computer Science and Technology, Zhejiang University of Technology, 288 Liuhe Road, Hangzhou 310023, China
[2]School of Mechanical Engineering, Shanghai Jiaotong University, 800 Dongchuan Road, Shanghai 200240, China
[3]School of Mechanical Engineering, Xi'an Jiaotong University, 28 West Xianning Road, Xi'an 710049, China
huyahong@zjut.edu.cn

Abstract To address the challenges associated with the inefficient generation of product functional design schemes in the face of complex user requirements, a multi-expert optimized reinforcement learning search network (MORN) is proposed. By adding non-functional factors to functional design knowledge representation, the algorithm breaks through the limitations of traditional single-dimensional evaluations and optimizes the generated functional unit chains. A highly efficient circular experience pool and dynamic priority sampling strategy are proposed to improve experience storage efficiency and training stability. Combining the dynamic weighting mechanism and the Mixture of Experts Model enhances the algorithm's adaptability to complex design tasks. Experiments show that the circular experience pool technology can eliminate memory fragmentation, reducing the model convergence steps and training time by 29.17%, and 88.19%, respectively. The dynamic weighting mechanism maintains a stable success rate of 93.60% in scenarios with variable requirements, and the MoE model increases the search success rate to 94.33%.

Keywords: functional knowledge integration, knowledge representation, deep reinforcement learning, Mixture of Experts.

1 Introduction

The growing complexity and diversity of user demands have increased challenges in product design. Relying solely on individual expertise and team experience often falls short in meeting rapidly evolving market needs. With the advancement of Internet technologies, a distributed design knowledge ecosystem has emerged, offering designers access to vast design resources. According to design science theory, new product design scheme comes from effective integration of existing knowledge. Knowledge integration (also termed design synthesis) refers to the systematic process of retrieving, selecting, and combining appropriate design knowledge to obtain novel products that fulfill

user requirements. Among various approaches, functional knowledge integration (FKI) is especially critical for generating high-quality conceptual solutions. This study focuses on FKI methodologies.

Fig. 1 illustrates a case of applying FKI to design a flat-plate solar water heater. The goal is to supply warm water using solar energy. Through FKI, three functional units (FU) FU1, FU2 and FU3 are selected from the distributed design knowledge base and integrated to realize the product's functional design.

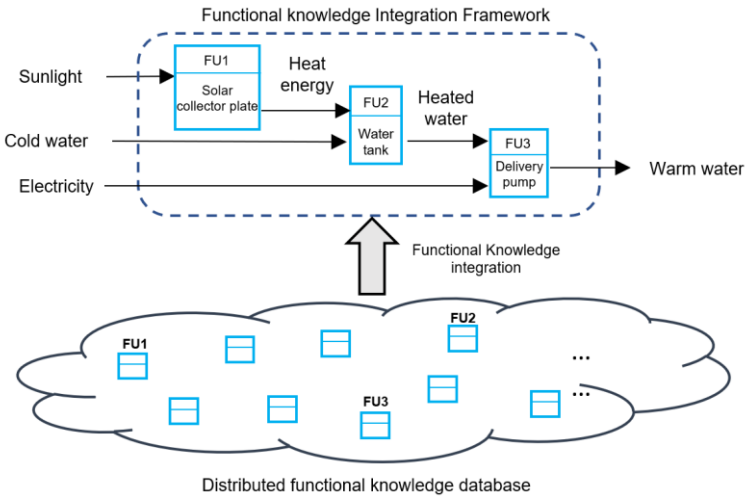

Fig. 1. An Example of FKI.

Current strategies for functional knowledge selection and integration predominantly rely on expertise of designers and simplistic rules, failing to fully leverage data-driven intelligent optimization methods. This limitation significantly constrains FKI's adaptability and generalization capabilities in complex environments. Furthermore, the diverse functionalities provided by different functional units lead to inconsistent representation formats across units. Establishing an effective functional knowledge representation model and developing intelligent FKI algorithms constitute critical prerequisites for realizing robust FKI.

This research proposes an AI-assisted FKI framework to address inefficiencies in navigating large-scale knowledge domains and better meet practical design needs. The main contributions include:

(1) Enhancing functional unit representation by incorporating non-functional characteristics;

(2) Optimizing the reward function and reinforcement learning network structure;

(3) Introducing a ring-structured experience replay mechanism to improve experience reuse and computational efficiency, thereby boosting learning performance and adaptability.

2 Related work

Functionality forms the core of product design, with designers focusing on fulfilling diverse user needs, including material, psychological, and social requirements during conceptualization [1–4]. To address this complexity, product functions are typically categorized into transformation, support, storage, and stimulation [2]. Flow-based input-output models have been widely adopted, such as Wang's [5] image-similarity-based flow matching using material, energy, and information flows. While effective for material needs, these models struggle with psychological and social requirements.

To improve FKI, Chen and Xie [6],[7] introduced a keyword-based representation using functional units as the smallest knowledge entities. They reformulated FKI as a multi-source path search problem, enabling automatic scheme generation. Enhancements like auxiliary unit chains and incomplete matching improved scalability in complex scenarios [8].

However, most existing integration approaches rely on exhaustive top-down or bottom-up searches. While effective for small knowledge bases, they become computationally infeasible as scale increases. Zhang et al. [9] addressed this by proposing fine-grained knowledge graph representations that preserve hierarchy and support efficient graph reasoning. To improve scalability, Chen et al. [10],[11] implemented parallel search across distributed processors. Zhang et al. [12] introduced a high-performance graph computing framework, and Yuan's team [13] advanced graph processing on multi-core CPU/GPU platforms.

Yet, these methods still suffer from combinatorial explosion. To address this, Lan et al. [14] proposed a reinforcement learning (RL) framework that models FKI as an exploration task, fundamentally eliminating exhaustive search.

Experience replay is crucial in RL for stabilizing learning by reusing past experiences. Traditional replay uses uniform sampling, while Schaul et al. [15] introduced Prioritized Experience Replay (PER) based on TD error to emphasize important samples. Cassirer et al. [16] developed the Reverb framework for improved replay efficiency. Buzzega et al. [17] applied replay to continual learning, and Li et al. [18] proposed AMPER for more efficient sampling in deep RL.

Replay buffer capacity significantly affects diversity and stability [19–21]. Pan et al. [22] proposed a value-function-based method that reduces memory usage while maintaining performance. Eren et al. [23] further evaluated buffer efficiency under constrained resources.

Despite AI's potential, traditional RL faces challenges in data dependency and poor generalization under limited memory. Existing FKI platforms often lack efficient storage and representation. To address these gaps, our work introduces a space-efficient replay architecture and an enhanced functional-unit model, boosting sample efficiency and generalization within memory constraints.

3 Problem Description

The functional knowledge unit integration algorithm operates within a reinforcement learning framework, where an intelligent agent explores optimal design paths on a functional knowledge graph, as shown in Fig. 2.

Experiences acquired through environmental interactions are stored in a replay buffer to support offline training of the agent model. A detailed exposition follows. To solve the FKI task, we model functional-unit chaining as an episodic decision process on the knowledge graph and train a Deep Monte Carlo Searching (DMCS) agent: in each episode the agent rolls out a path from the start to the target node, computes the episode return for each state–action pair, and updates a Q-network via Monte Carlo targets.

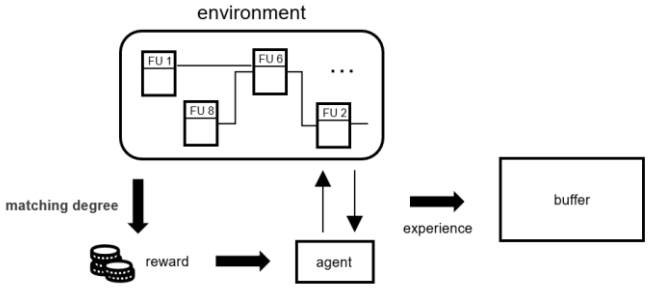

Fig. 2. Functional Unit Exploration Method.

3.1 The keyword representation method of functional units

The keyword-based method represents a functional unit by describing its inputs and output using structured keywords [6], [7]. As shown in Fig. 3, each FU can have multiple inputs and a single output. Keywords follow a "modifier + core word" format, where the modifier adds semantic detail to the core word. Fig. 4 provides an example.

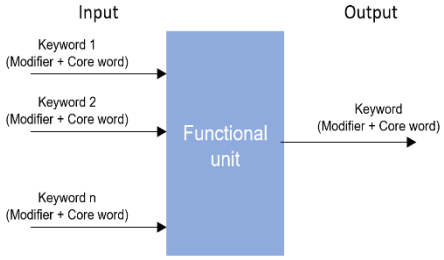

Fig. 3. Functional Unit Representation.

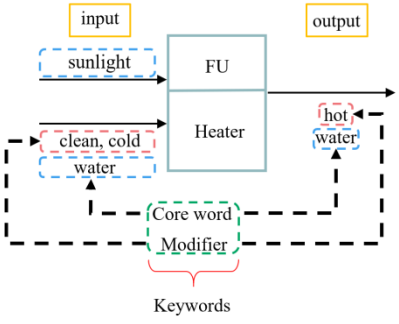

Fig. 4. Example of Input/Output Structure.

While effective, the current method has limitations. If two FUs share identical inputs and outputs, they are treated as equivalent despite differences in non-functional attributes such as cost, reliability, and lead time. While such attributes are critical in FU selection, they are overlooked by current descriptions, which prevents the optimal solution to be obtained.

3.2 Experience Replay Bottlenecks

Conventional deep reinforcement learning employs first-in-first-out (FIFO) replay to speed up experience storage and retrieval. However, in sparse-reward settings, FIFO overwrites valuable early experiences, regardless of their importance. For instance, in robotic grasping tasks24, only 0.3% of samples are successful, yet 50% of those are discarded within 10,000 training steps.

Additionally, traditional uniform sampling does not prioritize the experiences, causing slower convergence due to unweighted sampling. This issue reduces the effectiveness of experience replay in complex learning scenarios.

4 Proposed Methodology

4.1 Incorporation of Real-World Factors

To address the shortcomings of the current keyword representation of FU, this paper proposes to add non-functional characteristics to the keyword representation to enable the optimal integration process of functional units. Considering real-world enterprise requirements, three non-functional attributes are chosen, i.e., product cost, lead time, and reliability. The improved representation of a FU is shown in Fig. 5.

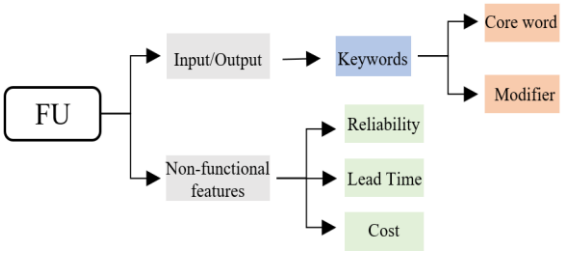

Fig. 5. Representation of Input/Output and Non-functional Features.

(1) Reliability: It is defined as the probability that *N* identical functional carriers remain operational after a specified time *t*. If *n(t)* units fail within this period, the reliability is calculated using Equation (1).

$$Reliability = 1 - \frac{n(t)}{N} \tag{1}$$

(2) Cost: Defined as the ex penses incurred by the supplier or manufacturer to implement the specified functional unit through the functional carrier.

(3) Lead Time: Defined as the time required by the supplier or manufacturer to deliver a product capable of implementing the specified functional unit to the customer.

4.2 Joint Metric Optimization of the Reinforcement Learning Model's Reward Function

To enhance practical applicability, we incorporate non-functional characteristics of functional units into the reward function. These characteristics allow customers to tailor solutions to specific needs and improve functional knowledge integration.

In our previous work14, the reinforcement learning model generated functional unit chains using a reward based solely on unit matching redundancy. The reward function is defined using Equation(2).

$$r = max(0,10 - Redundancy) \tag{2}$$

where "*Redundancy*" represents the redundancy between functional units, reflecting the degree of their mapping, and *r* represents the actual reward. A lower redundancy indicates higher complementarity between functional units, making them more suitable for functional chain generation. Therefore, the reward value increases as redundancy decreases. By applying an inverse incentive mechanism of 10 - *Redundancy*, functional units with strong complementarity are prioritized. The function *max*(0,·) is used to ensure that the reward value remains non-negative, preventing negative values caused by excessive redundancy. Additionally, the constant 10 is introduced to control the reward range, effectively distinguishing different redundancy levels while ensuring the stability and effectiveness of the reinforcement learning process.

To incorporate non-functional features into the functional knowledge integration process, these features are introduced as joint influence indicators in the reward calculation.

(1) The newly introduced three factors are normalized and weighted as the basis for the reward value. This approach eliminates the dimensional differences between indicators, ensuring that their values are unified within the range of 0 to 1. This not only facilitates comparison and subsequent factor management but also promotes network convergence.

(2) Given the heterogeneous customer prioritization among the factors, the reward function formulation necessitates implementing differential weighting coefficients. WF_r is the weight of reliability, WF_c, WF_l, and WF_{redn} are the weights for cost, lead time, and redundancy, respectively. To ensure that the reward remains within a reasonable and observable range, we keep the total success reward value *r* within the fixed range of 0 to 10.

(3) For the factors "cost" and "lead time," we know from real-world considerations that lower values result in higher product profitability, which is desirable for customers. Therefore, the variable $Value_{cost}$ is used instead of $Value$.

$$Value_{cost} = 1 - \frac{Cost}{Cost_{max}} \tag{3}$$

where $Cost$ represents the actual cost of the current design solution, measured in monetary units (e.g., RMB). $Cost_{max}$ denotes the maximum price threshold for similar products, determined jointly by market demand and budget constraints.

Similarly, $Value_{leadtime}$ is introduced.

$$Value_{leadtime} = 1 - \frac{LeadTime}{LeadTime_{max}} \tag{4}$$

where $LeadTime$ represents the actual cycle time from product design to delivery, measured in days. $LeadTime_{max}$ is the longest allowable delivery period stipulated in the customer's contract

(4) The calculation formula for r is as follows:

$$r = Reliability * WF_r + Value_{cost} * WF_c + Value_{leadtime} * WF_l + max(0,10 - Redundancy) * WF_{redn} \tag{5}$$

4.3 Efficient Circular Experience Replay Buffer Storage

Traditional linear memory storage often fragments memory and loses valuable historical data. To solve this, we use a circular buffer with pre-allocated memory blocks to enhance locality and eliminate the need for dynamic memory management. The storage structure is shown in Fig. 6.

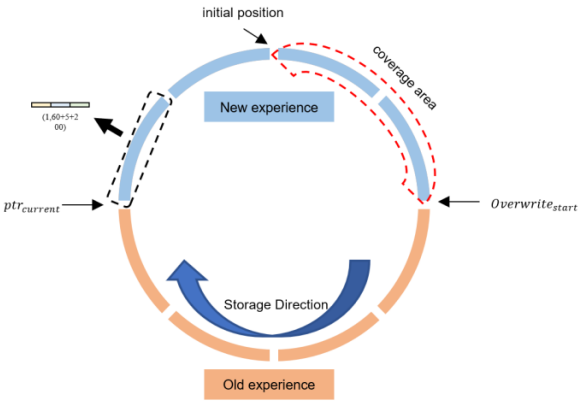

Fig. 6. Model Diagram Of Experience Buffer.

An experience is recorded as:

$$exp_{st} = [X_1^t, X_2^t, X_3^t] \tag{6}$$

where X_1^t, X_2^t, X_3^t, respectively, represent the experience groups formed by the state features, temporal trajectories, and values from the joint indicators.

When full, old data in the buffer are overwritten using FIFO, and this algorithm prevents fragmentation and ensures both new and historical data are preserved. The overwrite pointer is updated via:

$$Overwrite_{start} = (ptr_{current} + N_{new})\ mod\ N_{buffer} \quad (7)$$

where N_{new} is the number of newly added experiences, and N_{buffer} denotes the total buffer capacity. ptr_{current} represents the pointer position at the end of the last experience storage, while $Overwrite_{start}$ indicates the starting position for storing the next batch of experiences after inserting the new experiences into the experience buffer. "Initial position" represents the initial storage location of the valid space in memory.

The circular buffer stores not only recent but also a considerable amount of historical experiences, which achieve a balance between timeliness and diversity to boost performance in complex tasks.

4.4 Experience Replay Mechanism

To avoid the inefficiencies of uniform sampling and the temporal disruption introduced by Prioritized Experience Replay (PER), we propose a recency-aware sampling method during data overwriting. As shown in Fig. 7, each mixed sampling batch consists of experiences drawn from the most recent experiences in a proportion of $(1 - \rho)$ and uniformly sampled from the entire dataset in a proportion of ρ.

$$\mathcal{I} = Shuffle\left(\mathcal{U}[Overwrite_{start} - \lfloor \rho N_{buffer} \rfloor, Overwrite_{start}) \cup \mathcal{U}[0, N_{buffer})\right) \quad (8)$$

Here, $\rho \in [0,1]$ is the proportion of recent experiences considered. $\mathcal{U}$ denotes a uniformly sampled index set from the interval $[a,b)$, while $Overwrite_{start} - \lfloor \rho N_{buffer} \rfloor$ indicates the starting position of the recent experience segment. The Shuffle(·) denotes the index random permutation operation. Although both recent and global indices are randomly generated, direct merging can introduce local temporal correlations. A uniform random shuffle is applied to ensure temporally uniform sampling to avoid positional bias and meet the i.i.d. assumption required for stable gradient updates. Without this step, sampling order may introduce implicit patterns that affect model robustness.

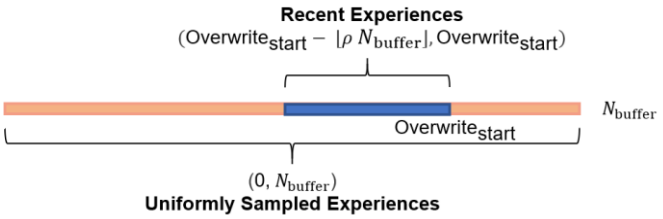

Fig. 7. Mixed Sampling Index Generation Method.

Experiments based on multi-factor joint indicators show that the circular buffer effectively retains experiences from the agent's exploration. A sufficiently large buffer avoids premature data deletion, helping the agent learn from both recent and historical experiences more effectively.

4.5 Functional Unit Chain Generation Algorithm

In this section, the deep reinforcement learning network to generate the functional unit chain is described in detail.

4.5.1 Representation of Functional Knowledge Graph

In the knowledge base, the relationships between functional units can be described using a graph model. Therefore, introducing the concept of a knowledge graph in this task is a natural choice, as such a graph database structure can support subsequent knowledge reasoning 14. FKG is modeled as a directed graph *G(V,E)*, where *V* represents the set of nodes in *G*, and *E* represents the set of edges. Each node $v_i \in V$ represents a functional unit, while each edge $e_{ij} \in E$ represents the relationship between functional units v_i and v_j.

4.5.2 Feature Network Expert Model

To enhance the adaptability and generalization ability of the network architecture in handling cross-domain design knowledge, this paper refines the model proposed in [14] and puts forward a Mixture of Experts (MoE) model. The core idea of the MoE model is to dynamically select the most suitable expert subnetworks based on the multi-feature information of the input data. Each expert models a distinct feature subspace, and diversity is encouraged through variations in structure and regularization. A gating network calculates a SoftMax-based weight distribution $g_i(x)$ to determine expert relevance. To reduce computation, only the top-k experts are activated for each input, and their outputs are combined as follows:

$$g_i(x) = \frac{exp\big(h_i(x)\big)}{\sum_{j=1}^{N} exp\left(h_j(x)\right)} \tag{9}$$

where $h_i(x)$ represents the raw score generated by the *i*-th expert through a linear transformation in the gating network, *N* represents the number of experts, and $g_i(x)$ denotes the weight of the *i*-th expert.

$$\tilde{g}_i(x) = \begin{cases} g_i(x) & if\ i \in Top_k\big(g_1(x), \cdots, g_N(x)\big) \\ 0 & otherwise \end{cases} \tag{10}$$

$$Features = \sum_{i=1}^{k} \tilde{g}_i(x) \cdot E_i(x) \tag{11}$$

where $E_i(x)$ represents the output of the *i-th* expert subnetwork, while $\tilde{g}_i(x)$ denotes its corresponding weight, and *k* refers to the number of selected expert subnetworks.

The model is jointly optimized using task loss and an entropy-based regularization term, which stabilizes the gating weight distribution. The objective function is:

$$L = L_{task} + \lambda \sum_{i=1}^{N} g_i(x)\, log\, g_i\,(x) \tag{12}$$

Here, *L* represents the total loss function, which is a weighted sum of the task loss and the regularization term. L_{task} denotes the task loss function, which measures the

prediction error of the model on the current task. It optimizes the parameters of both the expert subnetworks and the gating network, ensuring the accuracy of path scoring. λ is the regularization coefficient, and the logarithm function is used to penalize uneven weight distribution, preventing the gating network from assigning weights too concentrated. Entropy regularization is used to constrain the distribution of the original gating network weights $g_i(x)$.

4.5.3 Network Architecture Model

The input features of the network model comprise environmental state data and agent-observed behavioral information, including the current node, subsequent nodes, local redundancy between the current and subsequent nodes, action paths, global redundancy of the current path, and requirement of the product. As shown in Fig. 8, functional unit features are encoded as an $x \times y$ matrix, where x denotes the maximum number of inputs or outputs per unit and y is the maximum number of keywords per input/output. Each row corresponds to an input or output, with columns 1 to $y-1$ storing modifier word vectors and the last column storing the core word vector. Absent keywords are represented with zero vectors. To conserve memory, action trajectories are encoded as the mean of word vectors for each input's modifiers and core word. Keyword embeddings are 200-dimensional vectors derived from Tencent AI Lab's ChineseEmbedding dataset. In this experiment, the values of x and y are set to 12 and 6, respectively.

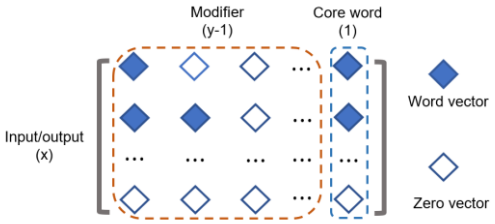

Fig. 8. Example of Encoding Matrix of Input and Output.

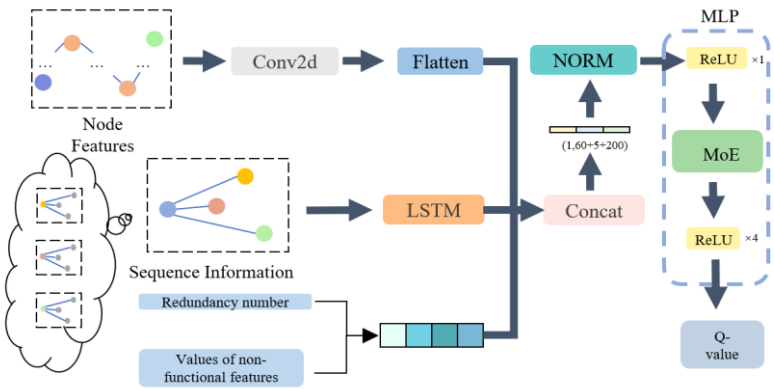

Fig.9. Overview of Network Architecture.

The specific network structure is shown in Fig. 9. It includes a 1×6 convolutional layer with 200 channels for feature extraction, followed by a single-layer unidirectional LSTM with 200 hidden units for sequential modeling. The outputs are concatenated with redundancy and component features to form the final representation.

The MoE model contains 4 expert networks, as shown in Fig.10. These experts learn the modeling capabilities of specific feature subspaces through gating weights, focusing on Node Feature Extraction, Multi-dimensional Joint Information Evaluation, Temporal Dependency of Time-sensitive paths, and Optimization Capability of Redundancy constrained path matching, respectively. According to Google's GShard architecture 25, with an activation ratio of 2:1, the expert model achieves an optimal trade-off between model capacity and computational efficiency. In the small-scale experiments used here, only 2 experts are activated for training selection, allowing for more suitable training based on specific input features. Since it is embedded in the fully connected layer, the input and output dimensions remain consistent.

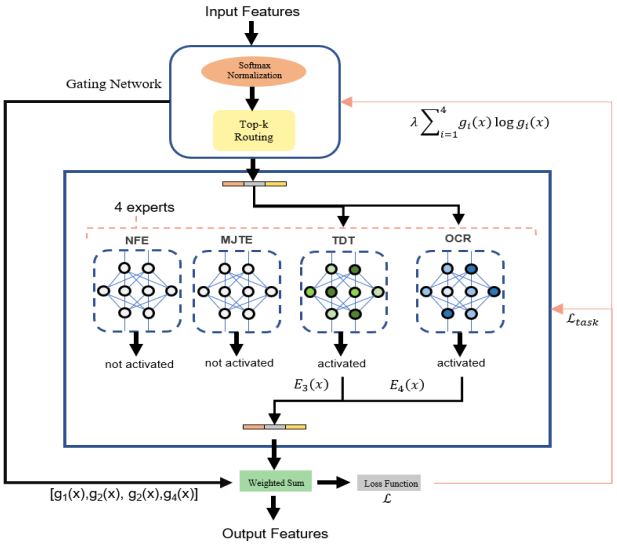

Fig. 10. Mixture of Experts.

4.5.4 Training algorithm

Complete network training algorithm is shown in Fig.11. The algorithm achieves efficient agent training through the multi-expert optimized reinforcement learning search network (MORN), and Mean Squared Error (MSE) is used as the loss function . First, it combines an interval-greedy strategy, using the hyperparameter ϵ to control the agent's exploration and exploitation. The agent interacts with the environment in each episode, generating state, action, and reward data, which are stored in the experience buffer. The reward is updated by calculating the Monte Carlo return to adjust future cumulative rewards, making the current reward more valuable in the long term. When the amount of experience in the buffer exceeds a set threshold, an experience replay mechanism is employed, and recent proportional priority sampling is used to extract

batch data to train the Q-network. A circular buffer strategy is designed, which cyclically overwrites old data and dynamically manages the experience pool.

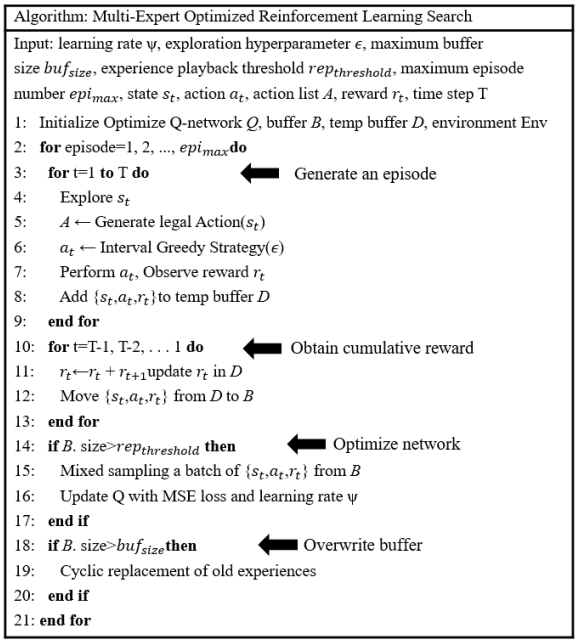

Fig. 11. Training Algorithm.

5 Experiments and analysis

This section presents experiments to evaluate the performance improvements of MORN. We compare MORN with the baseline algorithm DMCS, proposed by Lan [14], which leverages reinforcement learning for autonomous agent exploration. DMCS effectively mitigates gradient explosion and significantly reduces the search time for generating functional chains.

We begin with the experimental setup, followed by analysis of the results.

5.1 Data preparation

Due to the limited size and lack of standard metrics in existing datasets for functional knowledge units, we follow the same data augmentation strategy as Lan et al. [14]. Starting from 59 original functional units, 354 training samples and 1,000 validation samples are generated. For real-world evaluation, we further test the models on 21 actual product design cases to ensure robustness across scenarios.

5.2 Implementation details & training condition

MORN algorithm is based on the PyTorch deep learning framework, using the Adam optimizer and MSE loss function, ReLU activation for the MLP layers, and Layer Norm

normalization. It was trained on a server with 20 processors (Intel i7-12700KF CPU @ 3.61 GHz) and an NVIDIA GeForce RTX 3090Ti 24GB GPU. The training duration was 8.5 hours. The values for each hyperparameter are as follows: maximum search length L=10, learning rate φ = 5e −7, the number of episodes epi_{max} = 1, 500,000, batch size *bat* = 512, experience replay threshold $\text{rep}_{\text{threshold}}$ = 1536, and maximum buffer size *buf*= 60,000.

To simulate realistic integration scenarios, non-functional attributes were randomly assigned: reliability ∈ [0,1], cost ∈ [0,100], and lead time ∈ [0,10]. The loss throughout the training process is shown in Fig.12. At the beginning of training, the learning policy starts exploration from scratch, causing the loss curve to drop rapidly in a short period. Then, as the agent enters the exploration phase with frequent policy updates, the curve fluctuates significantly and exhibits an upward trend. As ϵ gradually decreases, the policy stabilizes, and the loss converges. The rewards during training are shown in Fig. 12 (b). The reward increases as ϵ decreases in certain regions. The overall reward trend first rises and then stabilizes, indicating that training in each region is sufficient. Eventually, the reward converges around 5.3.

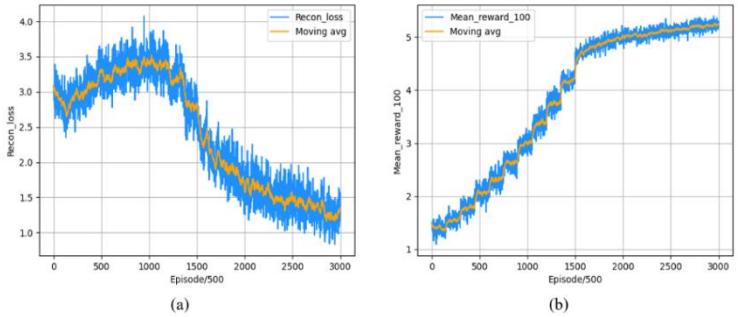

Fig. 12. Loss (a) and Mean Reward (b) Versus Training Episodes (per 500 Episodes).

5.3 Performance comparison experiments

We evaluate performance using 1,000 randomly generated functional requirements from Lan's demand generator [18]. The maximum search length is set to 10, and the search time is limited to 300 seconds.

We compare MORN and DMCS across two key dimensions, i.e., computational efficiency and result quality. The computational efficiency of the algorithm is evaluated using success, mean time, and total time. Success represents the probability of successfully finding the optimal path within the maximum search time. Mean time refers to the average search time required to find the optimal path. Total time denotes the total time taken to complete all search tasks. Result quality is the quality of the search results and it is measured using the mean redundancy (mean redn) metric, which indicates the average redundancy of the optimal paths.

In real-world scenarios, design requirements often differ in priority. To ensure a more balanced and reasonable product design, the influence of multiple joint factor indicators is averaged. Additionally, we conduct weighted experiments to reflect user-

specific preferences, assigning higher weights to redundancy due to its significance in design completeness. Other joint indicators are adjusted to remain consistent with the reward function in Lan's study, ensuring fairness and comparability.

5.3.1 Multi-Factor Combined Metrics (MFCM)

MFCM introduces four equally weighted evaluation metrics to refine reward. As shown in Table 1, MFCM improves the success rate by 4.70% with the cost of the increase of execution time.

Table 1. Comparison of Multi-Factor Combined Metrics (MFCM) Experiments.

	Success(%)	Mean redn	Mean time(s)	Total time (s)
DMCS	88.70	5.94	0.09	105.26
DMCS+MFCM	93.40	20.79	0.22	233.80

By replacing the coarse single redundancy metric (gradient 1.0) with fine-grained feedback (each metric contributing 0.025), the agent better perceives state transitions. This facilitates more accurate decisions and naturally promotes balanced redundancy among functional units.

5.3.2 High-Efficiency Ring-structured Experience Pool (HREP)

HREP enhances storage and training efficiency using preallocated memory and a FIFO overwrite strategy. Data is averaged over 30 independent experiments, with memory block distribution states sampled every 10 minutes and experience overwrite cycle duration calculated. The results shown in Table 2 indicate that memory fragmentation is completely eliminated, while overwrite latency is reduced by 86.59%. It also shortens training time from 72 to 8.5 hours and reduces convergence steps by 29.17%, as shown in Table 3.

Table 2. Storage Efficiency Comparison.

	DMCS (Linear Array)	HREP (Ring Buffer)	Improvement Ratio
Memory Fragmentation Rate (%)	32.72	0.00	100%
Overwrite Operation Latency (μs)	8.20	1.10	86.59%

Table 3. Training Efficiency Comparison.

	DMCS (Linear Array)	HREP (Ring Buffer)	Improvement Ratio
Convergence Steps (experience replays)	1.2×10^6	8.5×10^5	29.17%
Average Training Time (hours)	72.00	8.50	88.19%

As shown in Table 4, though HREP does not significantly outperform baselines on small datasets, its linear memory scalability makes it well-suited for large-scale or dynamic industrial scenarios.

Table 4. High-Efficiency Ring-structured Experience Pool Experiment Comparison.

	Success(%)	Mean redn	Mean time(s)	Total time (s)
DMCS	88.70	5.94	0.09	105.26
DMCS+HREP	89.53	5.50	0.09	105.18

5.3.3 Heterogeneous Factors (HF)

To accommodate varied customer priorities, dynamic weight adjustments were introduced:

$$[WF_r, WF_c, WF_l, WF_{redn}] = [1,3,2,4] \tag{13}$$

As shown in Table 5, the model maintained a 93.60% success rate under random weight shifts, which demonstrates robustness to heterogeneous design requirements.

Table 5. Dynamic Weight Experiment Comparison.

	Success(%)	Mean redn	Mean time(s)	Total time(s)
DMCS+MFCM	93.40	20.79	0.22	233.80
DMCS+MFCM(HF)	93.60	19.40	0.18	195.44

5.3.4 MoE Hybrid Expert Model (MoE)

As shown in Table 6, the MoE model introduces a dynamic gating mechanism for adaptive expert selection, achieving the highest success rate of 94.33% while reducing redundancy by 15.32%. Though it incurs a 20.74% increase in computational cost due to gating and parallel expert inference, response time remains under 0.25s per step, which is acceptable for industrial applications.

Table 6. Comparison of Hybrid Expert Network Experiments.

	Success(%)	Mean redn	Mean time(s)	Total time(s)
DMCS+MFCM+HREP	94.10	22.06	0.18	193.70
DMCS+MFCM+HREP+MoE	94.33	18.68	0.22	233.88

Overall, through progressive optimization across four stages, the success rate improved from 88.70% to 94.33%. This synergy supports dynamic user preferences and significantly improves efficiency compared to manual processes, offering a practical solution for intelligent product design.

6 Conclusion and Future Work

In this work, we optimized the functional knowledge integration framework DCMS to improve its performance. By refining the reward distribution, our approach ensures that design requirements are accurately captured, preventing deviations in product solutions due to mismatches within the knowledge base. Additionally, the adoption of a more efficient experience pool significantly enhances model training efficiency and enables the discovery of more optimal design solutions.Furthermore, we propose a MoE module to improve adaptability and generalization across diverse design tasks. For future work, we plan to explore multi-agent parallel search, where multiple agents collaborate to improve search efficiency. Given the confidentiality constraints of enterprise products, we will also investigate a distributed knowledge base framework to enable functional product integration without requiring a centralized knowledge repository.

Acknowledgement

This work is supported by Xie Youbai Design Science Foundation (No. XYB-DS-202205), National Key R&D Program of China (No. 2018YFB0204003), and National Natural Science Foundation of China (No. 62176237).

References

1. Gero J S. Design prototypes: a knowledge representation schema for design[J]. AI magazine, 1990, 11(4): 26-26.
2. Stone R B, Wood K L. Development of a functional basis for design[C]//International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. American Society of Mechanical Engineers, 1999, 19739: 261-275.
3. Tang L. An approach to function identification in automated conceptual design of mechanism systems[J]. Research in Engineering Design, 2008, 19: 151-159.
4. Kurtoglu T, Swantner A, Campbell M I. Automating the conceptual design process:“From black box to component selection”[J]. Ai Edam, 2010, 24(1): 49-62.
5. Wang H, Zhang P, Nie Z, et al. An intelligent integrated innovation design method based on flow functional genes coding and digitization[J]. Advanced Engineering Informatics, 2025, 64: 103044.
6. Chen B, Xie Y B. A computer-assisted automatic conceptual design system for the distributed multi-disciplinary resource environment[J]. Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science, 2017, 231(6): 1094-1112.
7. Chen B, Xie Y B. Functional knowledge integration of the design process[J]. Science China technological sciences, 2017, 60: 209-218.
8. Chen B, Xie Y B. A function unit integrating approach for the conceptual design synthesis in the distributed resource environment[J]. Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science, 2018, 232(5): 759-774.

9. Zhang Y, Wang H, Zhai X, et al. A C-RFBS model for the efficient construction and reuse of interpretable design knowledge records across knowledge networks[J]. Systems Science & Control Engineering, 2021, 9(1): 497-513.
10. Chen B, Hu J, Qi J, et al. Concurrent multi-process graph-based design component synthesis: Framework and algorithm[J]. Engineering Applications of Artificial Intelligence, 2021, 97: 104051.
11. Chen B, Hu J, Chen W, et al. Scalable multi-process inter-server collaborative design synthesis in the Internet distributed resource environment[J]. Advanced Engineering Informatics, 2021, 47: 101251.
12. Zhang Yu, Jiang Xinyu, Yu Hui, Zhao Jin, Qi Hao, Liao Xiaofei, Jin Hai, Wang Biao, Yu Ting. A Survey on Key Technologies of Graph Computing Architecture and System Software [J]. Journal of Computer Research and Development, 2024, 61(1): 20-42.
13. Zhang Yuan, Cao Huawei, Zhang Jie, Shen Yue, Sun Yiming, Dun Ming, An Xuejun, Ye Xiaochun. A Survey on Key Technologies of Graph Processing Systems for Multi-Core CPU and GPU Platforms [J]. Journal of Computer Research and Development, 2024, 61(6): 1401-1428.
14. Lan X, Hu Y, Xie Y, et al. Innovation design oriented functional knowledge integration framework based on reinforcement learning[J]. Advanced Engineering Informatics, 2023, 58: 102122.
15. Schaul T, Quan J, Antonoglou I, et al. Prioritized experience replay[J]. arXiv preprint arXiv:1511.05952, 2015.
16. Cassirer A, Barth-Maron G, Brevdo E, et al. Reverb: a framework for experience replay[J]. arXiv preprint arXiv:2102.04736, 2021.
17. Buzzega P, Boschini M, Porrello A, et al. Rethinking experience replay: a bag of tricks for continual learning[C]//2020 25th International Conference on Pattern Recognition (ICPR). IEEE, 2021: 2180-2187.
18. Li M, Kazemi A, Laguna A F, et al. Associative memory based experience replay for deep reinforcement learning[C]//Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design. 2022: 1-9.
19. Zhang S, Sutton R S. A deeper look at experience replay[J]. arXiv preprint arXiv:1712.01275, 2017.
20. Catto E. Box2d: A 2d physics engine for games. URL: http://www. box2d. org, 2011.
21. Bellemare M G, Naddaf Y, Veness J, et al. The arcade learning environment: An evaluation platform for general agents[J]. Journal of artificial intelligence research, 2013, 47: 253-279.
22. Lan Q, Pan Y, Luo J, et al. Memory-efficient reinforcement learning with value-based knowledge consolidation[J]. arXiv preprint arXiv:2205.10868, 2022.
23. Eren H A, Adar N, Yazar A. The Importance of Experience Replay Buffer Size in Deep Reinforcement Learning[C]//12th International Congress on Engineering, Architecture and Design, 2023.
24. Kalashnikov, D., Irpan, A., Pastor, P., et al. QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation[J]. arXiv preprint arXiv:1806.10293, 2018.
25. Lepikhin D, Lee H J, Xu Y, et al. Gshard: Scaling giant models with conditional computation and automatic sharding[J]. arXiv preprint arXiv:2006.16668, 2020.