AutoMC: Automated Model Compression Based on Domain Knowledge and Progressive Search

1st Chunnan Wang  
Harbin Institute of Technology  
Harbin, China  
WangChunnan@hit.edu.cn

2nd Hongzhi Wang* Member, IEEE  
Harbin Institute of Technology  
Harbin, China  
wangzh@hit.edu.cn

3rd Xiangyu Shi  
Harbin Institute of Technology  
Harbin, China  
xyu.shi@hit.edu.cn

Abstract—Model compression methods can reduce model complexity on the premise of maintaining acceptable performance, and thus promote the application of deep neural networks under resource constrained environments. Despite their great success, the selection of suitable compression methods and design of details of the compression scheme are difficult, requiring lots of domain knowledge as support, which is not friendly to non-expert users. To make more users easily access to the model compression scheme that best meet their needs, in this paper, we propose AutoMC, an effective and efficient automatic tool for model compression. In order to improve the search efficiency and quality, in AutoMC, we build the domain knowledge on model compression to deeply understand the characteristics and advantages of each compression method under different settings. This method can provide AutoMC with the more reasonable guidance and thus reduce useless evaluation. In addition, we present a progressive search strategy to efficiently explore pareto optimal compression scheme according to the learned prior knowledge combined with the historical evaluation information. This strategy can help AutoMC selectively and gradually explore more valuable search space, and thus reduce the search difficulty and improve the search efficiency. Extensive experimental results show that AutoMC can provide users with better compression schemes within short time compared to the existing compression methods and AutoML algorithms, which demonstrates the effectiveness and significance of our proposed algorithm.

Index Terms—Automated machine learning, model compression, domain knowledge, progressive search

I. INTRODUCTION

Neural networks are very powerful and can handle many real-world tasks. They have many applications on data engineering, such as feature engineering [1], [2], [3], [4], graph classification [5], spatio-temporal data mining [6], graph mining [7], [8] and more. Despite their great success, parameter amounts of neural networks are generally very large, bringing expensive computation and storage cost. In order to apply them to mobile devices building more intelligent mobile devices, many model compression methods have been proposed, including model pruning [9]–[13], knowledge distillation [14], low rank approximation [12], [15] and so on.

These compression methods can effectively reduce model parameters while maintaining model accuracy as much as possible, but are difficult to use. Each method has many hyperparameters that can affect its compression effect, and different methods may suit for different compression tasks. Even the domain experts need lots of time to test and analyze for designing a reasonable compression scheme for a given compression task. This brings great challenges to the practical application of compression techniques.

In order to enable ordinary users to easily and effectively use existing model compression techniques, in this paper, we propose AutoMC, an Automatic Machine Learning (AutoML) algorithm to help users automatically design model compression schemes. Note that in AutoMC, we do not limit a compression scheme to only use a compression method under a specific setting. Instead, we allow different compression methods and methods under different hyperparameters settings to work together (execute sequentially) to obtain diversified compression schemes. We try to integrate advantages of different methods/settings through this sequential combination so as to obtain more powerful compression effect, and our experimental results prove this idea to be effective and feasible.

However, the search space of AutoMC is huge. The number of compression strategies contained in the compression scheme may be of any size, which brings great challenges to the subsequent search tasks. In order to improve the search efficiency, we present following two innovations to improve the performance of AutoMC from the perspectives of knowledge introduction and search space reduction, respectively.

Specifically, for the first innovation, we built domain knowledge on model compression, which discloses technical and settings details of compression strategies, and their performance under some common compression tasks. This domain knowledge can assist AutoMC to deeply understand potential characteristics and advantages of each component in the search space. It guides AutoMC select more appropriate compression strategies to build effective compression schemes, and thus reduce useless evaluation and improve the search efficiency.

As for the second innovation, we adopted the idea of progressive search space expansion to improve the search efficiency of AutoMC. Specifically, in each round of optimization, we only take the next operations, i.e., unexplored next-step compression strategies, of the evaluated compression scheme as the search space. Then, we select the pareto optimal operations for scheme evaluation, and finally take the next

1In this paper, a compression strategy refers to a compression method with a specific hyperparameter setting.
operations of the new scheme as the newly expanded search area to participate in the next round of optimization. In this way, AutoMC can selectively and gradually explore more valuable search space, reduce the search difficulty, and improve the search efficiency. In addition, AutoMC can analyze and compare the impact of subsequent operations on the performance of each compression scheme in a fine-grained manner, and finalize a more valuable next-step exploration route for implementation, thereby effectively reducing the evaluation of useless schemes.

The final experimental results show that AutoMC can quickly search for powerful model compression schemes. Compared with the existing AutoML algorithms which are non-progressive and ignore domain knowledge, AutoMC is more suitable for dealing with the automatic model compression problem where search space is huge and components are complete and executable algorithms.

Our contributions are summarized as follows:

1. **Automation.** AutoMC can automatically design the effective model compression scheme according to the user demands. As far as we know, this is the first automatic model compression tool.

2. **Innovation.** In order to improve the search efficiency of AutoMC algorithm, an effective analysis method based on knowledge base and a progressive search strategy are designed. As far as we know, AutoMC is the first AutoML algorithm that introduce external knowledge.

3. **Effectiveness.** Extensive experimental results show that with the help of knowledge base and progressive search strategy, AutoMC can efficiently search the optimal model compression scheme for users, outperforming compression methods designed by humans.

The remainder of this paper is organized into six sections. Section II introduces the existing model compression approaches and the AutoML techniques. Section III introduces in details the core techniques involved in our proposed AutoMC algorithm. In Section IV and Section V, we respectively give the overall framework of the AutoMC algorithm and analyze its consistency. Finally, Section VI conducts extensive experiments to examine the effectiveness of AutoMC. Finally, we draw conclusions and present the future work in Section VII.

II. RELATED WORK

In this section, we introduce the existing model compression approaches and the AutoML techniques, which are involved in this paper.

A. Compression Methods

People utilize compression methods to address the escalating demands for memory efficiency in various deep learning applications. One prevailing direction is the compression of data storage and memory usage. Researchers have proposed many effective compression methods toward different sub-directions, e.g. compression of distributed communication cost [16], compression of memory cost [17], compression of data [18], [19], and etc.

In this paper we mainly discuss compression for the model itself. By reducing the parameters of the model, it is possible to applying large neural networks to mobile or embedding devices, which has been widely studied in the community. Researchers have proposed many effective compression methods, and they can be roughly divided into the following four categories. (1) pruning methods, which aim to remove redundant parts e.g., filters, channels, kernels or layers, from the neural network [20]–[23]; (2) knowledge distillation methods that train the compact and computationally efficient neural model with the supervision from well-trained larger models; (3) low-rank approximation methods that split the convolutional matrices into small ones using decomposition techniques [24]; (4) quantization methods that reduce the precision of parameter values of the neural network [25], [26].

These compression methods have their own advantages, and have achieved great success in many compression tasks, but are difficult to apply as is discussed in the introduction part. We aim to flexibly use the experience provided by them to support the automatic design of model compression schemes.

Numerous model compression systems have been proposed to optimize model compression from a systemic perspective, such as SparTA [27], BMCook [28], SparseTIR [29], Flash-LLM [30], and etc. The difference between them and AutoMC is that AutoMC is an AutoML framework, which can employ several compression methods to perform one compression task together. For other compression systems, they can only employ limited number of compression criteria to compress the model, such as pruning or quantization, which cannot combine various compression criteria into one compression task. Our work deals with the compression task in a higher level enabling to merge advantages of different compression methods to gain better performance. Our work demonstrates the combination of various compression methods can achieve better performance than single compression method.

B. Automated Machine Learning Algorithms

The goal of Automated Machine Learning (AutoML) is to realize the progressive automation of ML, including automatic design of neural network architecture, ML workflow [31], [32] and automatic setting of hyperparameters of ML model [33], [34]. The idea of the existing AutoML algorithms is to define an effective search space which contains a variety of solutions, then design an efficient search strategy to quickly find the best ML solution from the search space, and finally take the best solution as the final output.

Search strategy has a great impact on the performance of the AutoML algorithm. The existing AutoML search strategies can be divided into 3 categories: Reinforcement Learning (RL) methods [35], Evolutionary Algorithm (EA) based methods [36], [37] and gradient-based methods [38], [39]. The RL-based methods use a recurrent network as controller to determine a sequence of operators, thus construct the ML solution sequentially. EA-based methods initialize a population of ML solutions first and then evolve them with their validation accuracies as fitnesses. As for the gradient-based methods,
they are designed for neural architecture search problems. They relax the search space to be continuous, so that the architecture can be optimized with respect to its validation performance by gradient descent [40]. They fail to deal with the search space composed of executable compression strategies. Therefore, in this paper, we only compare AutoMC’s search strategy with the previous two methods.

III. OUR APPROACH

We firstly introduce the overall framework of AutoMC algorithm (Section III-A), and then give the related concepts on model compression and problem definition of automatic model compression (Section III-B). Then, we make full use of the existing experience to construct an efficient search space for the compression area (Section III-C). Finally, we designed a search strategy, which improves the search efficiency from the perspectives of knowledge introduction and search space reduction, to help users quickly search for the optimal compression scheme (Section III-D).

A. Overview of AutoMC Algorithm

AutoMC can be divided into two parts: effective embedding learning and progressive search strategy.

In the embedding learning part (Section III-C), AutoMC introduces domain knowledge, including related knowledge graph and experimental experiences, to learn characteristics of compression strategies in the search space. This method provides AutoMC with more reasonable guidance and thus reduce useless evaluation. In this way, AutoMC can make more accurate decisions under the guidance of higher-quality embeddings.

As for the progressive search part (Section III-D), AutoMC constructs a progressive search space, exploring more valuable subspace in each iteration. Also, it screens out more promising compression strategies from the subspace for evaluation. The progressive search strategy designed in AutoMC can help AutoMC selectively and gradually explore more valuable search space, reduce the search difficulty, and improve the search efficiency. The overall framework of AutoMC is shown in Fig. 4.

B. Related Concepts and Problem Definition

Related Concepts. Given a neural model $M$, we use $P(M)$, $F(M)$ and $A(M)$ to denote its parameter amount, FLOPS and its accuracy score on the given dataset, respectively. Given a model compression scheme $S = \{s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_k\}$, where $s_i$ is a compression strategy ($k$ compression strategies in the scheme are required to be executed in sequence), we use $S[M]$ to denote the compressed model obtained after applying scheme $S$ to the model $M$. In addition, we use $R(S, M) = \frac{A(S[M]) - A(M)}{A(M)} \in [0, 1]$, where $\ast$ can be $P$ or $F$, to represent model $M$’s reduction rate on parameter amount or FLOPS after executing scheme $S$. We use $AR(S, M) = \frac{A(S[M]) - A(M)}{A(M)} > -1$ to represent accuracy increase rate achieved by $S$ on $M$.

III. OUR APPROACH

We firstly introduce the overall framework of AutoMC algorithm (Section III-A), and then give the related concepts on model compression and problem definition of automatic model compression (Section III-B). Then, we make full use of the existing experience to construct an efficient search space for the compression area (Section III-C). Finally, we designed a search strategy, which improves the search efficiency from the perspectives of knowledge introduction and search space reduction, to help users quickly search for the optimal compression scheme (Section III-D).

A. Overview of AutoMC Algorithm

AutoMC can be divided into two parts: effective embedding learning and progressive search strategy.

In the embedding learning part (Section III-C), AutoMC introduces domain knowledge, including related knowledge graph and experimental experiences, to learn characteristics of compression strategies in the search space. This method provides AutoMC with more reasonable guidance and thus reduce useless evaluation. In this way, AutoMC can make more accurate decisions under the guidance of higher-quality embeddings.

As for the progressive search part (Section III-D), AutoMC constructs a progressive search space, exploring more valuable subspace in each iteration. Also, it screens out more promising compression strategies from the subspace for evaluation. The progressive search strategy designed in AutoMC can help AutoMC selectively and gradually explore more valuable search space, reduce the search difficulty, and improve the search efficiency. The overall framework of AutoMC is shown in Fig. 4.

B. Related Concepts and Problem Definition

Related Concepts. Given a neural model $M$, we use $P(M)$, $F(M)$ and $A(M)$ to denote its parameter amount, FLOPS and its accuracy score on the given dataset, respectively. Given a model compression scheme $S = \{s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_k\}$, where $s_i$ is a compression strategy ($k$ compression strategies in the scheme are required to be executed in sequence), we use $S[M]$ to denote the compressed model obtained after applying scheme $S$ to the model $M$. In addition, we use $R(S, M) = \frac{A(S[M]) - A(M)}{A(M)} \in [0, 1]$, where $\ast$ can be $P$ or $F$, to represent model $M$’s reduction rate on parameter amount or FLOPS after executing scheme $S$. We use $AR(S, M) = \frac{A(S[M]) - A(M)}{A(M)} > -1$ to represent accuracy increase rate achieved by $S$ on $M$.

In this paper, a compression method denotes an existing model compression work, and a compression strategy refers to a compression method with a specific hyperparameter setting. As for the model compression scheme, it is a sequence of compression strategies.

Definition 1 (Automatic Model Compression). Given a neural model $M$, a target reduction rate of parameters $\gamma$ and a search space $S$ on the model compression schemes, the Automatic Model Compression problem aims to quickly find the optimal compression scheme $S^* \in S$:

$$S^* = \arg\max_{S \in S} f(S, M) \quad \text{such that} \quad PR(S, M) \geq \gamma$$

\[ f(S, M) := [AR(S, M), PR(S, M)] \]

A Pareto optimal compression scheme that performs well on two optimization objectives: $PR$ and $AR$, and meets the target reduction rate of parameters.

In this paper, we intend to design an effective search space with a variety of compression schemes and efficient search strategies to effectively solve this Automatic Model Compression problem.

C. Search Space on Compression Schemes

In AutoMC, we utilize some open source model compression methods to build a search space on model compression. Specifically, we collect 6 effective model compression methods, allowing them to be combined flexibly to obtain diverse model compression schemes to cope with different compression tasks. In addition, considering that hyperparameters have great impact on the performance of each method, we regard the compression method under different hyperparameter settings as different compression strategies, and intend to find the best compression strategy sequence, that is, the compression scheme, to effectively solve the actual compression problems.

Table I gives these compression methods. These methods and their respective hyperparameters constitute a total of 9,252 compression strategies. Utilizing these compression strategies to form compression strategy sequences of different lengths (length $< L$), then we get a search space $S$ with $\sum_{l=0}^{L}(9252)^l$ different compression schemes.

Our search space $S$ can be described as a tree structure (as is shown in Fig. 1), where each node (layer $\leq L$) has 9,252 child nodes corresponding to 9,252 compression strategies and
nodes at layer $L + 1$ are leaf nodes. In this tree structure, each path from START node to any node in the tree corresponds to a compression strategy sequence, namely a compression scheme in the search space.

D. Search Strategy of AutoMC Algorithm

The search space $S$ is huge. In order to improve the search performance, we introduce domain knowledge to help AutoMC learn characteristics of components of $S$ (Section III-D1). In addition, we design a progressive search strategy to finely analyze the impact of subsequent operations on the compression scheme, and thus improve search efficiency (Section III-D2).

1) Domain Knowledge based Embedding Learning: We build a knowledge graph on compression strategies, and extract experimental experience from the related research papers to learn potential advantages and effective representation of each compression strategy in the search space. Considering that two kinds of knowledge are of different types$^2$ and are suitable for different analytical methods, we design different embedding learning methods for them, and combine two methods for better understanding of different compression strategies.

Knowledge Graph based Embedding Learning. We build a knowledge graph $G$ that exposes the technical and settings details of each compression strategy, to help AutoMC to learn relations and differences between different compression strategies. $G$ contains five types of entity nodes: $(E_1)$ compression strategy, $(E_2)$ compression method, $(E_3)$ hyperparameter, $(E_4)$ hyperparameter’s setting and $(E_5)$ compression technique type. Also, it includes five types of entity relations:

- $R_1$: the corresponding relation between a compression strategy and its compression method ($E_1 \rightarrow E_2$)
- $R_2$: the corresponding relation between a compression strategy and its hyperparameter setting ($E_1 \rightarrow E_4$)
- $R_3$: the corresponding relation between a compression method and its hyperparameter ($E_2 \rightarrow E_3$)
- $R_4$: the corresponding relation between a compression method and its compression technique type ($E_2 \rightarrow E_5$)
- $R_5$: the corresponding relation between a hyperparameter and its setting ($E_3 \rightarrow E_4$)

$^2$knowledge graph is relational knowledge whereas experimental experience belongs to numerical knowledge.
Algorithm 1 Compression Strategy Embedding Learning

Input: \( C \): Compression strategies in Table I; \( G \): Knowledge graph on \( C \); \( E \): Experiment experience w.r.t. \( G \) from papers involved in Table I; \( \text{TrainEpoch} \): Maximum training epochs

1: \textbf{while} epoch < \( \text{TrainEpoch} \) \textbf{do}
2: \hspace{1em} Execute one epoch training of TransR using triplets in \( G \)
3: \hspace{1em} \( e_{C,P_{i,j}} \leftarrow \) Extract knowledge embedding of compression strategy \( C_iP_{i,j} \) \((\forall C_iP_{i,j} \in C)\)
4: \hspace{1em} Optimize the obtained knowledge embedding using \( E \) according to Equation 3
5: \hspace{1em} \( \tilde{e}_{C,P_{i,j}} \leftarrow \) Extract the enhanced embedding of \( C_iP_{i,j} \) \((\forall C_iP_{i,j} \in C)\)
6: \hspace{1em} Replace \( e_{C,P_{i,j}} \) by \( \tilde{e}_{C,P_{i,j}} \) \((\forall C_iP_{i,j} \in C)\)
7: \textbf{end while}
8: \textbf{return} \hspace{1em} High-level embedding of compression strategies:
\[ \tilde{e}_{C,P_{i,j}} \hspace{1em} (\forall C_iP_{i,j} \in C) \]

\( R_1 \) and \( R_2 \) describe the composition details of compression strategies, \( R_3 \) and \( R_4 \) provide a brief description of compression methods, \( R_5 \) illustrate the meaning of hyperparameter settings. Fig. 2 (a) is an example of \( G \).

We use TransR [43] to effectively parameterize entities and relations in \( G \) as vector representations, while preserving the graph structure of \( G \). Specifically, given a triplet \((h, r, t)\) in \( G \), we learn embedding of each entity and relation by optimizing the translation principle:

\[ W_r e_h + e_r \approx W_t e_t \]  \hspace{1em} (2)

where \( e_h, e_t \in \mathbb{R}^d \) and \( e_r \in \mathbb{R}^k \) are the embedding for \( h, t, \) and \( r \) respectively; \( W_r \in \mathbb{R}^{k \times d} \) is the transformation matrix of relation \( r \).

This embedding learning method can inject the knowledge in \( G \) into representations of compression strategies, so as to learn effective representations of compression strategies. In AutoMC, we denote the embedding of compression strategy \( C_iP_{i,j} \) learned from \( G \) by \( e_{C_iP_{i,j}} \).

Experimental Experience based Embedding Enhancement. Research papers contain many valuable experimental experiences: the performance of compression strategies under a variety of compression tasks. These experiences are helpful for deeply understanding performance characteristics of each compression strategy. If we can integrate them into embeddings of compression strategies, then AutoMC can make more accurate decisions under the guidance of higher-quality embeddings.

Based on this idea, we design a neural network, which is denoted by \( \mathcal{N} \mathcal{N}_{\text{exp}} \) (as shown in Fig. 2 (b)), to further optimize the embeddings of compression strategies learned from \( G \). \( \mathcal{N} \mathcal{N}_{\text{exp}} \) takes \( e_{C_iP_{i,j}} \) and the feature vector of a compression task \( \text{Task}_k \) (denoted by \( e_{\text{Task}_k} \)) as input, intending to output \( C_iP_{i,j} \)’s compression performance, including parameter’s reduction rate \( PR \), and accuracy’s increase rate.

\[ \min_{\theta, e_{C_iP_{i,j}}(C_iP_{i,j}) \in C} \frac{1}{|E|} \sum_{(C_iP_{i,j}, \text{Task}_k, AR, PR) \in E} \left( \| \mathcal{N} \mathcal{N}_{\text{exp}}(e_{C_iP_{i,j}}, \text{Task}_k; \theta) - (AR, PR) \| \right) \]  \hspace{1em} (3)

where \( \theta \) indicates the parameters of \( \mathcal{N} \mathcal{N}_{\text{exp}} \), \( C \) represents the set of compression strategies in Table I, and \( E \) is the set of experimental experience extracted from papers.

Pseudo Code. Combining the above two learning methods, then AutoMC can comprehensively consider knowledge graph and experimental experience and obtain a more effective embeddings. Algorithm 1 gives the complete pseudo code of the embedding learning part of AutoMC.

2) Progressive Search Strategy: Taking the compression scheme as the unit to analyze and evaluate during the search phase can be very inefficient, since the compression scheme evaluation can be very expensive when its sequence is long. The search strategy may cost much time on evaluation while only obtain less performance information for optimization, which is ineffective.

To improve search efficiency, we apply the idea of progressive search strategy instead in AutoMC. We try to gradually add the valuable compression strategy to the evaluated compression schemes by analyzing rich procedural information, i.e., the impact of each compression strategy on the original compression strategy sequence, so as to quickly find better schemes from the huge search space \( S \).
Algorithm 2 Progressive Search Strategy

**Input:** Initial compression scheme \( H_{\text{scheme}} = \{\text{START}\} \);
Initial optimal compression schemes \( \text{OPT}_{\text{START}} = \emptyset \);
High-level embedding of compression strategies \( \tilde{e}_{C,P,i,j} \) (\( \forall C,P,i,j \in \mathbb{C} \)); Search epochs \( \text{SearchEpoch} \); Target reduction rate of parameters \( \gamma \)
1: while epoch < \text{SearchEpoch} do
2: \( H_{\text{sub}} \) ← Sample some schemes from \( H_{\text{scheme}} \)
3: \( S_{\text{step}} \leftarrow \{(seq, s) \mid \forall seq \in H_{\text{sub}}, s \in \text{Next seq}\} \)
4: \( \text{ParetoO} \leftarrow \{\text{argmax}_{(seq,s)} \in S_{\text{step}}[\text{ACC}_{seq,s}, \text{PAR}_{seq,s}]\} \)
5: Evaluate schemes in \( \text{ParetoO} \) and get \( \text{PAR}_{seq,s}^{step} \).
6: Optimize the weights \( \omega \) of multi-objective evaluator \( F_{\text{mo}} \) according to Equation 5 (using the embedding of compression strategies \( \tilde{e}_{C,P,i,j} \))
7: \( H_{\text{scheme}} \leftarrow H_{\text{scheme}} \cup \{(seq^*, s^*) \mid (seq^*, s^*) \in \text{ParetoO}\} \)
8: \( \text{OPT}_{seq} \leftarrow \text{OPT}_{seq} - \{s\} \), \( \text{OPT}_{seq} \leftarrow \emptyset \) for each \( (seq^*, s^*) \in \text{ParetoO} \)
9: \( \text{ParetoSchemes} \leftarrow \text{Pareto} \) optimal compression schemes with parameter decline rate \( \geq \gamma \) in \( H_{\text{scheme}} \)
10: end while
11: return \( \text{ParetoSchemes} \)

Algorithm 3 Overall Framework of AutoMC

**Input:** \( \mathbb{C} \): Compression strategies in Table I; \( \tilde{e} \): Knowledge graph on \( \mathbb{C} \); \( \mathbb{E} \): Experiment experience w.r.t. \( \mathbb{G} \) from papers involved in Table I; \( \text{TrainEpoch} \): Maximum training epochs; \( \text{SearchEpoch} \): Search epochs; \( \gamma \): Target reduction rate of parameters
1: \( H_{\text{scheme}} \leftarrow \{\text{START}\}, \text{OPT}_{\text{START}} \leftarrow \emptyset \)
2: \( \tilde{e}_{C,P,i,j} \leftarrow \) Algorithm 1(\( \mathbb{C} \), \( \tilde{e} \), \( \mathbb{E} \), \( \text{TrainEpoch} \))
3: \( \text{ParetoSchemes} \leftarrow \) Algorithm 2(\( H_{\text{scheme}}, \text{OPT}_{\text{START}}, \tilde{e}_{C,P,i,j}, \text{SearchEpoch}, \gamma \))
4: return \( \text{ParetoSchemes} \)

Specifically, we propose to utilize historical procedural information to learn a multi-objective evaluator \( F_{\text{mo}} \) (as shown in Fig. 3). We use \( F_{\text{mo}} \) to analyze the impact of a newly added compression strategy \( s_{t+1} = C_{i,j} \in \mathbb{C} \) on the performance of compression scheme \( seq = (s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_t) \), including the accuracy improvement rate \( \text{AR}_{\text{step}} \) and reduction rate of parameters \( \text{PR}_{\text{step}} \).

For each round of optimization, we firstly sample some Pareto-Optimal and evaluated schemes \( seq \in H_{\text{scheme}} \), take their next-step compression strategies \( \text{Next seq} \subseteq \mathbb{C} \) as the search space \( S_{\text{step}}: s_{\text{step}} \leftarrow \{(seq, s) \mid \forall seq \in H_{\text{scheme}}, s \in \text{Next seq}\} \), where \( H_{\text{scheme}} \subseteq \mathbb{H} \) scheme are the sampled schemes. Secondly, use \( F_{\text{mo}} \) to select pareto optimal options \( \text{ParetoO} \) from \( S_{\text{step}} \), thus obtain better compression schemes \( seq^* \rightarrow s^*, \forall (seq^*, s^*) \in \text{ParetoO} \) for evaluation.

\[
\text{ParetoO} = \{\text{argmax} \left[ \text{ACC}_{seq,s}, \text{PAR}_{seq,s} \right] \}_{(seq,s) \in S_{\text{step}}}
\]
\[
\text{ACC}_{seq,s} = (\text{seq}[M]) \times (1 + \text{AR}_{\text{step}})
\]
\[
\text{PAR}_{seq,s} = (\text{seq}[M]) \times (1 - \text{PR}_{\text{step}})
\]

where \( \text{AR}_{\text{step}} \) and \( \text{PR}_{\text{step}} \) are performance changes that \( s \) brings to scheme \( seq \) predicted by \( F_{\text{mo}} \). \( \text{ACC}_{seq,s} \) and \( \text{PAR}_{seq,s} \) are accuracy and parameter amount obtained after executing scheme \( seq \rightarrow s \) to the original model \( M \).

Finally, we evaluate compression schemes in \( \text{ParetoO} \) and get their real performance changes, which are denoted by \( \text{AR}_{\text{seq,s}}^*, \text{PR}_{\text{seq,s}}^* \), and use the following formula to further optimize the performance of \( F_{\text{mo}} \):

\[
\min_{\omega} \frac{1}{|\text{ParetoO}|} \sum_{(seq,s) \in \text{ParetoO}} \left\| F_{\text{mo}}(seq^*, s^*; \omega) - (\text{AR}_{\text{seq,s}}^*, \text{PR}_{\text{seq,s}}^*) \right\|
\]

We add the new scheme \( \{seq^* \rightarrow s^* \mid (seq^*, s^*) \in \text{ParetoO}\} \) to \( H_{\text{scheme}} \) to participate in the next round of optimization steps.

### Advantages of Progressive Search
In this way, AutoMC can obtain more training data for strategy optimization, and can selectively explore more valuable search space, thus improve the search efficiency.

### IV. AUTOMC ALGORITHM

This section introduces the overall framework of AutoMC algorithm (Section IV-A), and then analyzes the time complexity of AutoMC algorithm (Section IV-B). Finally, we discuss the generalization of AutoMC (Section IV-C).

#### A. Overall Algorithm Framework

The overall pseudo code of AutoMC is shown in Algorithm 3. AutoMC firstly learns high-level embeddings of each compression strategy (Algorithm 1), and then uses the learned embeddings to represent compression strategies and previous strategy sequences that need to input to \( F_{\text{mo}} \) (Algorithm 2). Finally, AutoMC outputs the searched Pareto optimal compression schemes.

#### B. Complexity Analysis

The main time cost of Algorithm 1 is consumed by the following three parts: (1) Training TransR: Suppose the average complexity of training TransR using Triples is \( O(C_{\text{transr}}) \). The complexity of this step is \( O(C_{\text{transr}}) \). (2) Extracting (enhanced) knowledge embedding: This step involves iterating through each compression strategies. We denote the number of compression strategies as \( N_c \). The complexity of this step is \( O(N_c) \). (3) Optimizing knowledge embedding: Suppose the average complexity of optimizing knowledge embedding is \( O(C_{\text{optKG}}) \). The complexity of this step is \( O(C_{\text{optKG}}) \). Therefore, considering that the outer loop is
executed $TrainEpoch$ times, the overall complexity of Algorithm 1 is $O(TrainEpoch \times (C_{\text{transr}} + C_{\text{optKG}} + N_c))$.

The time cost of Algorithm 2 is mainly consumed by the following parts: (1) Generating $S_{\text{step}}$: This step involves iterating through each sub-scheme and computing the next possible step. Suppose the average size for a sub-scheme is $K$, and the size of $Next_{\text{seq}}$ is $N_{\text{next}}$. The complexity of this step is $O(KN_{\text{next}})$. (2) Evaluating schemes in ParetoOpt: This step involves evaluating each scheme in ParetoOpt. Suppose the size of ParetoOpt is $N_{\text{pareto}}$, and the average complexity of evaluating one compression strategy is $C_{\text{eval}}$. The complexity of this step is $O(C_{\text{eval}}N_{\text{pareto}})$. (3) Optimizing weights $\omega$: Optimizing the weights $\omega$ is based on a multi-objective evaluator and depends on the optimization algorithm used. Suppose the average complexity of Optimization is $O(C_{\text{optim}})$. The complexity of this step is $C_{\text{optim}}$. Therefore, considering the above three steps, and the outer loop is executed $SearchEpoch$ times, the overall complexity of Algorithm 2 is $O(SearchEpoch \times (KN_{\text{next}} + C_{\text{eval}}N_{\text{pareto}} + C_{\text{optim}}))$.

The complexity of Algorithm 3 is the summation of Algorithm 1 and 2, i.e. $O(TrainEpoch \times (C_{\text{transr}} + C_{\text{optKG}} + N_c) + SearchEpoch \times (KN_{\text{next}} + C_{\text{eval}}N_{\text{pareto}} + C_{\text{optim}}))$.

C. Discussion of Generalization

AutoMC serves as a general framework for compression scheme searching, which can be easily decoupled with the specific compression task. It’s important to note that AutoMC, particularly its innovative approach involving domain knowledge and the progressive search strategy, can readily extend to other domains, such as Natural Language Processing (NLP). This adaptability involves selecting different compression methods and tailored search spaces for the target domain.

V. PROOF OF THE ALGORITHM CONSISTENCY

This section provides the proof of the consistency to learn a multi-objective evaluator $F_{\text{mo}}$ by utilizing historical procedural information.

We firstly split evaluator $F_{\text{mo}}$ into two evaluators, $F_{\text{mo1}}$ and $F_{\text{mo2}}$. The optimization formula of evaluator $F_{\text{mo1}}$ is:

$$\min_{\omega} \frac{1}{|ParetoOpt|} \sum_{(seq',s') \in ParetoOpt} ||F_{\text{mo1}}(seq', s'; \omega) - AR_{\text{step}} ||$$  \hspace{1cm} (6)

where ParetoOpt denotes the Pareto Optimal compression schemes found by the search strategy, and the optimization formula of evaluator $F_{\text{mo2}}$ is:

$$\min_{\omega} \frac{1}{|ParetoOpt|} \sum_{(seq',s') \in ParetoOpt} ||F_{\text{mo2}}(seq', s'; \omega) - PR_{\text{step}} ||$$  \hspace{1cm} (7)

Since the two evaluators are symmetrical, we will only discuss the consistency of evaluator $F_{\text{mo1}}$.

We give the definition of the empirical loss that an architecture is sampled and the expected loss on the entire data space as follows.

**Definition 1 (Empirical Loss).** The weighted empirical loss of a given evaluator $F_{\text{mo1}}$ is:

$$L(F_{\text{mo1}}) = \frac{1}{|ParetoOpt|} \sum_{(seq',s') \in ParetoOpt} ||F_{\text{mo1}}(seq', s'; \omega) - AR_{\text{step}} ||$$  \hspace{1cm} (8)

**Definition 2 (Expected Loss).** Let $S$ be an architecture search space, $D = \{(seq', s') | (seq', s') \sim H\}$ be a data space, the expected loss of a evaluator $F_{\text{mo1}}$ on the data space $D$ is:

$$LE(F_{\text{mo1}}) = \mathbb{E}_{(seq', s') \sim D} ||F_{\text{mo1}}(seq', s'; \omega) - AR_{\text{step}} ||$$  \hspace{1cm} (9)

We demand that the gap, $|L(F_{\text{mo1}}) - LE(F_{\text{mo1}})|$, between the empirical loss and the expected loss of evaluator $F_{\text{mo1}}$ is...
Theorem 1 (Azuma’s Inequality). Suppose \( \{X_k, k = 0, 1, 2, \ldots \} \) is a martingale and \( |X_k - X_{k-1}| < c_k \) almost surely. Then for all positive integers \( N \) and all positive reals \( \epsilon \):

\[
P \left[ \left| X_N - X_0 \right| \geq \epsilon \right] \leq 2 \exp \left( -\frac{\epsilon^2}{2 \sum_{k=1}^N c_k^2} \right)
\]  

(11)

where \( \omega \) is a concentration inequality of random process called Azuma’s Inequality (Azuma, 1967) can be used.

Theorem 2. Let \( \Psi \) be a hypothesis class containing all the possible hypotheses of evaluator \( \mathcal{F}_{\text{mol}} \). For any \( \delta > 0 \), with probability at least \( 1 - \delta \), \( \forall \Psi \in \Psi \):

\[
|L(\mathcal{F}_{\text{mol}}) - L_E(\mathcal{F}_{\text{mol}})| < \sqrt{2(d + \ln \frac{2}{\delta})} \frac{\mu}{|\text{ParetoO}|} 
\]  

(12)

where \( d \) is the Pollard’s pseudo-dimension of \( \Psi \).

Proof. To use Azuma’s inequality, we need our target sequence to be a martingale. We turn \((\text{seq}^t, s^t)\) into \((\text{seq}^t_1, s^t_1)\) based on the time sequence of \( \text{seq}^t \) entering \( \mathcal{H}_{\text{scheme}} \), where \( 1 \leq t \leq |\text{ParetoO}| \). We consider a sequence of random variables \( U_1, \ldots, U_{|\text{ParetoO}|} \), with \( U_i = \| \mathcal{F}_{\text{mol}}(\text{seq}^t_1, s^t_1; \omega) - \mathcal{F}_{\text{step}}(\text{seq}^t_1, s^t_1; \omega) \| - L_E(\mathcal{F}_{\text{mol}}) \). Since \( \mathcal{F}_{\text{step}}(\text{seq}^t_1, s^t_1; \omega) \in [0, 1] \) and \( \mathcal{F}_{\text{mol}}(\text{seq}^t_1, s^t_1; \omega) \in [0, 1] \), we have \( \|U_i\| \leq 1 \). Letting \( Z_t = \sum_{i=1}^t U_i \) and \( Z_0 = 0 \), \( Z_t \) is a martingale. For any \( 1 \leq t \leq |\text{ParetoO}| \):

\[
\mathbb{E}[Z_t | Z_{t-1}, \ldots, Z_0] = \mathbb{E}[U_t + Z_{t-1} | Z_{t-1}, \ldots, Z_0] = Z_{t-1}
\]  

(13)

We can apply Azuma’s inequality on \( Z_t \). Given \( |Z_t - Z_{t-1}| = |U_t| \leq 1 \) and \( |Z_{|\text{ParetoO}|} - Z_0| = |Z_{|\text{ParetoO}|}| = |\text{ParetoO}||(L(\mathcal{F}_{\text{mol}}) - L_E(\mathcal{F}_{\text{mol}}))| \), for any real value \( \lambda > 0 \), we have:

\[
P \left[ L(\mathcal{F}_{\text{mol}}) - L_E(\mathcal{F}_{\text{mol}}) \geq \frac{\lambda}{\sqrt{|\text{ParetoO}|}} \right] \leq 2e^{-\lambda^2/2}
\]  

(14)

Setting \( \lambda = \sqrt{2(d + \ln \frac{2}{\delta})} \), we can have the desired result in Theorem 2.

VI. EXPERIMENTS

In this part, we examine the performance of AutoMC. We firstly compare AutoMC with human designed compression methods to analyze AutoMC’s application value and the rationality of its search space design (Section VI-B). Secondly, we compare AutoMC with classical AutoML algorithms to test the effectiveness of its search strategy (Section VI-C). Then, we transfer the compression scheme searched by AutoMC to other neural models to examine its transferability (Section VI-D). Finally, we conduct ablation studies to analyze the impact of embedded learning method based on domain knowledge and progressive search strategy on the overall performance of AutoMC (Section VI-E).

We implemented all algorithms using Pytorch and performed all experiments using RTX 3090 GPUs.

A. Experimental Setup

Compared Algorithms. We compare AutoMC with two popular search strategies for AutoML:

- GraphNAS [44], an RL search strategy that combines recurrent neural network controller.
- RENAS [36], an EA-based search strategy for multi-objective optimization.
- Random, a commonly used baseline in AutoML.

To enable these AutoML algorithms to cope with our automatic model compression problem, we set their search space to \( \mathcal{S} \) \((L = 5)\). In addition, we employ 6 state-of-the-art human-invented compression methods:

- LMA [14]: A compression method using a highly efficient multi-segment activation which can rapidly produce multiple linear regions with very few parameters by leveraging the statistical information.
- LeGR [9]: LeGR learns a global ranking of the filters across different layers of convolutional networks, which is used to obtain a set of architectures that have different accuracy/latency trade-offs by pruning the bottom-ranked filters.
- NS [10]: During training, insignificant channels are automatically identified and pruned afterwards by NS, yielding thin and compact models with comparable accuracy.
- SFP [11]: SFP enables the pruned filters to be updated when training the model after pruning, which accelerates the inference procedure of convolutional neural networks.
- HOS [12]: Filters with low cumulant values that do not deviate significantly from Gaussian distribution are identified and removed by HOS to accelerate the network.
- LFB [15]: LFB reduces the number of parameters of convolutional networks by learning a basis of the filters in convolutional layers. For the forward pass, the learned basis is used to approximate the original filters and then used as parameters for the convolutional layers.

We use these compression methods as baselines, to show the importance of automatic model compression.

Compression Tasks. We construct two experiments to examine the performance of AutoML algorithms:

- Exp1: \( D = \text{CIAFR-10}, M = \text{ResNet-56}, \gamma = 0.3 \)
- Exp2: \( D = \text{CIAFR-100}, M = \text{VGG-16}, \gamma = 0.3 \)

where CIAFR-10 and CIAFR-100 [45] are two commonly used image classification datasets, and ResNet-56 and VGG-16 are two popular CNN network architectures.

To improve the execution speed, we sample 10% data from \( D \) to execute AutoML algorithms in the experiments. After
executing AutoML algorithms, we select the Pareto optimal compression scheme with \( PR \geq \gamma \) for evaluation. As for the existing compression methods, we apply grid search to get their optimal hyperparameter settings and set their parameter reduction rate to 0.4 and 0.7 to analyze their compression performance.

Furthermore, to evaluate the transferability of compression schemes searched by AutoML algorithms, we design two transfer experiments. We transfer compression schemes searched on ResNet-56 to ResNet-20 and ResNet-164, and transfer schemes from VGG-16 to VGG-13 and VGG-19.

**Implementation Details.** In AutoMC, the embedding size is set to 32. \( N_{\text{eeg}} \) and \( F_{\text{hmc}} \) are trained with the Adam with a learning rate of 0.001. After AutoMC searches for 3 GPU days, we choose Pareto optimal compression schemes as the final output. As for compared AutoML algorithms, we follow implementation details reported in their papers, and control running time of each AutoML algorithm to be the same. Fig. 5 gives the best compression schemes searched by AutoMC.

**Description of the Compression Schemes Searched.**

Fig. 5 illustrates the best compression schemes searched by AutoMC. In a compression scheme, compression methods are arranged in a sequence. A CNN model is sequentially sent to compression methods according to this sequence. The model output by the last compression method is our target model. Different compression methods are illustrated by frame with different colors. Blue frames represent compressed models, and data in the frames is formalized as PR(%) / FR(%) / Acc.(%). Just like the red part in the figure, additional fine-tuning will be added to the end of sequence to make up fine-tuning epoch for comparison, if the fine-tuning epoch of the last compression method is not enough.

We can observe that the model accuracy is not always decreasing during the compression process. Taking compression process with ResNet-56 on CIFAR-10 (PR = 70%) for example, the model accuracy increases after the 2nd and
4th compression method, while the number of parameter decreases. The result breaks the law that the larger the compression rate decreases, the more accuracy loss when a single compression method is used. The result also demonstrates the necessity of analyzing from multiple angles and performing fine-grained compression.

B. Comparison with the Performance Methods

Table II gives the performance of AutoMC and the existing compression methods on different tasks. We can observe that compression schemes designed by AutoMC surpass the manually designed schemes in all tasks. These results prove that AutoMC has great application value. It has the ability to help users search for better compression schemes automatically to solve specific compression tasks.

In addition, the experimental results show us: (1) A compression strategy may perform better with smaller parameter reduction rate ($PR$). Taking result of ResNet-56 on CIFAR-10 using LeGR as an example, when the $PR$ is 0.4, on average, the model performance falls by 0.0088% for every 1% fall in parameter amount; however, when $PR$ becomes larger, the model performance falls by 0.0737% for every 1% fall in parameter amount. (2) Different compression strategies may be appropriate for different compression tasks. For example, LeGR performs better than HOS when the $PR = 0.4$ whereas HOS outperforms LeGR when $PR = 0.7$. Based on the above two points, combination of multiple compression strategies and fine-grained compression for a given compression task may achieve better results. This is consistent with our idea of designing the AutoMC search space, and it further proves the rationality of the AutoMC search space design.

C. Comparison with the NAS algorithms

Table II gives the performance of different AutoML algorithms on different compression tasks. Fig. 6 provides the performance of the best compression scheme (Pareto optimal scheme with highest accuracy score) and all Pareto optimal schemes searched by AutoML algorithms.

a) Performance Analysis: We can observe that RL algorithm performs well in the very early stage, but its performance improvement is far behind other AutoML algorithms in the later stage. Evolution algorithm outperforms the other algorithms except AutoMC in both experiments. As for the Random algorithm, its performance has been rising throughout the entire process, but still worse than most algorithms.

b) Efficiency Analysis: According to Fig. 6 (a) and (c), compared with the existing AutoML algorithms, AutoMC can search for better model compression schemes more quickly, e.g. in Fig. 6 (a), in around 300 min, AutoMC has achieved 97% of the entire process, but still worse than most algorithms.
AutoMC to obtain a better compression scheme quickly, which proves its effectiveness and further proves the rationality of the AutoMC search strategy design.

D. Transfer Study

Table III, IV shows the performance of different models transferred from ResNet-56 and VGG-16. We can observe that LFB outperforms AutoMC with ResNet-20 on CIFAR-10. We believe that the reason is that LFB has a talent for dealing with small models. It’s obvious that the performance of LFB gradually decreases as the scale of the model increases. For example, LFB achieves an accuracy of 91.57% with ResNet-20 on CIFAR-10, but only achieves 24.17% with ResNet-164 on CIFAR-10. Except that, compression schemes designed by AutoMC surpass the manually designed schemes in all tasks. These results prove that AutoMC has great transferability. It is able to help users search for better compression schemes automatically with models of different scales.

Besides, the experimental results show that the same compression strategies may achieve different performance on models of different scales. In addition to the example of LFB and AutoMC above, LeGR performs better than HOS when using ResNet-20 whereas HOS outperforms LeGR when using ResNet-164. Based the above, combination of multiple compression strategies and fine-grained compression for models of different scales may achieve more stable and competitive performance.

E. Ablation Study

We further investigate the effect of the knowledge based embedding learning method, experience based embedding learning method and the progressive search strategy, three core components of our algorithm, on the performance of AutoMC using the following four variants of AutoMC, thus verify innovations presented in this paper.

1) AutoMC-KG. This version of AutoMC removes knowledge graph embedding method.

2) AutoMC-NN$_{exp}$. This version of AutoMC removes experimental experience based embedding method.

3) AutoMC-Multiple Source. This version of AutoMC only uses strategies w.r.t. LeGR to construct search space.

4) AutoMC-Progressive Search. This version of AutoMC replaces the progressive search strategy with the RL based search strategy that combines recurrent neural network.

Corresponding results are shown in Fig. 7, we can see that AutoMC has much better performance than AutoMC-KG and AutoMC-NN$_{exp}$, which ignore the knowledge graph or experimental experience on compression strategies while learning their embedding. This result shows us the significance and necessity of fully considering two kinds of knowledge on compression strategies in the AutoMC, for effective embedding learning. Our proposed knowledge graph embedding method can explore the differences and linkages between compression strategies in the search space, and the experimental experience based embedding method can reveal the performance characteristics of compression strategies. Two embedding learning methods can complement each other and help AutoMC have a better and more comprehensive understanding of search space components.

Also, We notice that AutoMC-Multiple Source achieve worse performance than AutoMC. AutoMC-Multiple use only one compression method to complete compression tasks. While AutoMC can increase the diversity of search space by introducing multiple compression methods and integrate the advantages of these compression methods. The result indicates the importance of using multi-source compression strategies to build the search space.

Besides, we observe that AutoMC-Progressive Search performs much worse than AutoMC, which proves that targeted search space expansion is conductive to search with a large-scale search space. RL’s unprogressive search process, i.e., only search for, evaluate, and analyze complete compression schemes, performs worse in the automatic compression scheme design problem task. It fails to effectively use historical evaluation details to improve the search effect and thus be less effective than AutoMC.

F. Analysis of Searched Pareto Optimal

To further analyze the compression schemes searched by AutoMC, we count the occurrence number of each compression method and its corresponding hyperparameter combination in the Pareto optimal schemes searched by AutoMC on the result of Exp1. Fig. 8 shows the occurrence number of each compression method and its corresponding hyperparameter combination in the Pareto optimal schemes searched by AutoMC on the result of Exp1.
Fig. 7. Pareto optimal results search by different versions of AutoMC algorithm on two automatic model compression tasks: Exp1 and Exp2 which are defined in Section VI-A. The two illustrations on the left were achieved by ResNet-56 on CIFAR-10, and the right by VGG-16 on CIFAR-100.

For SFP (C4), we observe that the counts of hyperparameter combination have a large difference. This result indicates that LeGR (C2) can be effectively used within different hyperparameter combinations, which means it can adapt to different compression tasks in the whole compression procedure by adjusting its hyperparameters. However, SFP (C4) is more sensitive to hyperparameter combination, which means it is more suitable for specific compression tasks.

Suggestions for the search space design: According to the analysis result of Fig. 8, further optimization of the search space can be conducted, such as removing LMA (C1) and LFB (C6) from the search space. The analysis of Fig. 9 indicates that further optimization can be conducted by removing redundant hyperparameter combinations of SFP (C4).

VII. CONCLUSION

In this paper, we propose the AutoMC to automatically design optimal compression schemes according to the requirements of users. AutoMC innovatively introduces external knowledge to assist search strategy to deeply understand the potential characteristics and advantages of each compression strategy, so as to design compression scheme more reasonably and easily. In addition, AutoMC presents the idea of progressive search space expansion, which can selectively explore valuable search regions and gradually improve the quality of the searched scheme through finer-grained analysis, for reducing the useless evaluations and improve the search efficiency. Extensive experimental results show that the combination of existing compression methods can create more powerful compression schemes, and the above two innovations make AutoMC more efficient than existing AutoML methods. In future work, we will transfer AutoMC into NLP domain, by considering more related compression methods into the search space. Furthermore, under the guidance of our experiment results, more optimization of our current search space can be performed to accelerate the search of AutoMC.

ACKNOWLEDGMENT

This work is supported by the National Key Research and Development Program of China (2021YFB3300502); National Natural Science Foundation of China (NSFC) (62232005, 6220212); ZTE Industry-University-Institute Cooperation Funds under Grant No. 2023ZTE09-02.
[43] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, “Learning entity and relation embeddings for knowledge graph completion,” in AAAI, B. Bonet and S. Koenig, Eds., 2015, pp. 2181–2187.

[44] Y. Gao, H. Yang, P. Zhang, C. Zhou, and Y. Hu, “Graph neural architecture search,” in IJCAI, C. Bessiere, Ed. ijcai.org, 2020, pp. 1403–1409.

[45] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” Handbook of Systemic Autoimmune Diseases, vol. 1, no. 4, 2009.