IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

IEEE RELIABILITY SOCIETY SECTION

Received January 16, 2022, accepted February 12, 2022, date of publication April 11, 2022, date of current version April 18, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3164510

TBEM: Testing-Based GPU-Memory Consumption Estimation for Deep Learning

HAIYI LIU[1], SHAOYING LIU[1], (Fellow, IEEE), CHENGLONG WEN[2], AND W. ERIC WONG[3], (Senior Member, IEEE)

[1]Department of Advanced Science and Engineering, Hiroshima University, Hiroshima 739-8511, Japan
[2]Microsoft Software Technology Center Asia, Microsoft, Suzhou 215123, China
[3]Department of Computer Science, The University of Texas at Dallas, Richardson, TX 75080, USA

Corresponding author: Shaoying Liu (sliu@hiroshima-u.ac.jp)

This work was supported by the Research Organization of Information and Systems (ROSI), National Institute of Informatics (NII) Open Collaborative Research 2021-(21FS02).

ABSTRACT Deep Learning (DL) has been successfully implemented and deployed to various software service applications. During the training process of DL, a large amount of GPU computing resources is required, but it is difficult for developers to accurately calculate the GPU resources that the model may consume before running, which brings great inconvenience to the development of DL systems. Especially in today's cloud-based model training. Therefore, it is very important to estimate the GPU memory resources that the DL model may use in a certain computing framework. Existing work has focused on static analysis methods to assess GPU memory consumption, highly coupled with the framework, and lack of research on low-coupled GPU memory consumption of the framework. In this article, we propose TBEM, which is a test-based method for estimating the memory usage of the DL model. First, TBEM generates enough DL models using an orthogonal array testing strategy and a classical neural network design pattern. Then, TBEM generates DL model tested in a real environment to obtain the real-time GPU memory usage values corresponding to the model. After obtaining the data of different models and corresponding GPU usage values, the data is analyzed by regression.

INDEX TERMS Deep learning, program static analysis, automated testing, bug detection.

I. INTRODUCTION

In recent years, with the improvement of computer performance and the continuous accumulation of data, the research and engineering implementation of artificial intelligence algorithms have made rapid progress. Among them, deep learning module is the most applied and implemented system in artificial intelligence system. It is widely used in many scenes, such as image recognition, speech recognition, recommendation system and so on. Although the accuracy and breadth of artificial intelligence system are improving year by year, the hardware cost and time cost of constructing neural network system are also increasing year by year. In 2020, the Gpt-3 model [1] published by Open Artificial Intelligence has 175 billion parameters, and the cost of network training is as high as 12 million US dollars. The high cost of model training is a common phenomenon of the neural network system. Facing such a high cost of model training, how to estimate the amount of memory that a deep learning model will occupy and ensure that the model does not out of memory during the training phase has become an important issue. This error is caused by the fact that developers cannot accurately estimate the size of the video memory occupied by the model before the model runs, so they cannot find the upper and lower limits of the super parameters suitable for their own development environment. According to relevant research literature, among all program failures of deep learning jobs, out of memory(OOM) account for 9.1% (including GPU and CPU) [2], and often occur in training process [3], which makes all the previous efforts of ongoing model training wasted. This not only wastes GPU computing resources, but also affects the development progress of engineers. Therefore, the memory consumption of different deep learning models and various deep learning libraries becomes particularly important. In terms of deep learning model and memory consumption, many researchers have made great contributions and provided corresponding solutions from different angles. The main methods include memory exchange, memory sharing, recalculation, and compressed neural network, etc. these methods reduce the

The associate editor coordinating the review of this manuscript and approving it for publication was Zhaojun Steven Li.

use of memory in the training process of deep learning model by analyzing the calculation graph model and using the technologies such as liveness analysis in static analysis or dynamic memory sharing and memory exchange. But their technology is usually used to make the built model input a larger batch size in the current hardware environment. Not to evaluate that the built model will cause memory overflow in a certain environment before model training.

In terms of deep learning framework and memory consumption, Gao *et al.* [4] proposed the method of using static analysis and calculation diagram and resident buffer to predict the memory utilization before model training.

Although the above methods have made effective solutions, there are still the following problems in the memory consumption evaluation of deep learning model:

- Deep learning library (e.g., TensorFlow, Pytoch) [5] generally contains two main functions, automatic differentiation, and GPU acceleration. Automatic differentiation is usually implemented by deep learning library, while GPU accelerated process is usually implemented by calling multiple NVIDIA components(e.g., CUDA, cudnn), it is difficult to achieve static analysis for the cooperative calls of multiple non-open-source components. Because the components called by the framework are in the closed source state, users cannot carry out common memory analysis methods such as context analysis. It also makes the deep learning library a black box for users.
- Each framework of deep learning is iterating rapidly in months, and new deep learning frameworks emerge one after another. The method of static analysis requires experts to analyze the framework. Therefore, the static analysis method undoubtedly increases the labor cost and time cost of evaluating the memory consumption of the deep learning model [6].

To solve the above problems, this paper proposes a method based on the combination of static analysis and dynamic test modeling analysis [7], [8]. Firstly, using the method of static analysis, the calculation graph of neural network is statically analyzed to pre-estimate the memory that may be consumed by the model. Then the pre-estimated model is run in the deep learning framework to obtain the real value of the model under the framework. Finally, Polynomial regression [9] is used to analyze the gap between the memory consumption estimated by static analysis and that of the real model, to deduce the possible memory consumption of the deep learning framework under different models.

II. PRELIMINARIES

In this section, we introduce three preliminaries used in TBEM, which are orthogonal array testing strategy, regression algorithm, and formal specification. The relationship between the above concepts and TBEM will be discussed in the overview section.

A. ORTHOGONAL ARRAY GENERATION METHOD

Orthogonal array generation method(OAGM), also known as Taguchi method, is a technology to generate orthogonal array(OA). The shape of the test case table depends on the number of factors and levels in the test [7].

Definition 1: An Orthogonal array can be defined as $OA(n, f, l, s)$, where:

- n is the number of rows of an orthogonal array. In an orthogonal array, n is known as *runs*.
- f indicates how many parameters (factors) need to be tested. In an orthogonal array, f is known as *factors*.
- l represents the value range of each parameter. In an orthogonal array, l is known as *levels*.
- s represents the *strength* of the orthogonal array. Let the orthogonal array be an $n \times f$ matrix A. In any $n \times s$ submatrix in A, There are $w = l \cdot d$ possible d-tuple rows, each of which appears the same number of times.

In Definition 1, *factors* correspond to hyperparameters in the deep learning framework that we need to test. *Levels* represent the range within which hyperparameters can be set. The number of *runs* is usually determined by *strength*. The relationship between specific parameters is as follows

- Orthogonal arrays are usually written as the following pattern: $L_{runs}(levels^{factors})$
- The value of *runs* is equal to $levels^{strength}$ when the levels of each factor are equal.
- When the number of *levels* in each *factor* of the orthogonal array is different, runs is equal to the product of the number of *levels* in the last *strength* column of the orthogonal array.

B. POLYNOMIAL REGRESSION

Definition 2: we have a polynomial equation of degree n represented as:

$$Y = \delta_0 + \delta_1 x_i + \delta_2 x_i^2 + \cdots + \delta_n x_i^n + \epsilon (i = 1, 2, \ldots, m)$$

can be expressed in matrix form in terms of a design matrix X, a response vector $\vec{y}$,a parameter vector $\vec{\delta}$ and a vector$\vec{\epsilon}$ of random errors. The i-th row of X and $\vec{y}$ will contain the x and y value for the i-th data sample,Then the model can be written as a system of linear equations:

$$\vec{y} = X\vec{\delta} + \vec{\epsilon}$$

The vector of estimated polynomial regression coefficients is:

$$\vec{\delta} = (X^T X)^{-1} X^T \vec{y}$$

assuming $m < n$ which is required for the matrix to be invertible, then since X is a Vandermonde matrix, the invertibility condition is guaranteed to hold if all the x_i values are distinct. This is the unique least-squares solution.

C. FORMAL SPECIFICATION

In order to select test cases generated by TBEM, ensure that the final test cases generated can be recognized by

the neural network framework. We need to investigate and summarize some neural network formal specifications. SOFL (Structured Object-Oriented Formal Language) as one of the Formal Engineering Methods for industrial software development [10]. In this paper, we use SOFL to write the formal specification of neural network. The reason is that the formal specification written by SOFL is easier for developers to understand and implement than the written by formal methods.

In SOFL, the operation of filtering test cases that do not conform to the specification can be represented by process, where **process** and **end_process** is a pair of keywords used to mark the beginning and end of the process. **pre** is a keyword indicating the start of the precondition of the process, and the keyword **post** indicates the start of the postcondition. Record a process as S, and then record pre-condition and post-condition as S_{pre} and S_{post} respectively. If the input variable of a process S meets S_{pre}, according to the specification, the output variable defined based on the input variable must meet S_{post} after S. Then we can have the following definitions:

Definition 3: Let $S_{post} \equiv (C_1 \wedge D_1) \vee (C_2 \wedge D_2) \cdots \vee (C_n \wedge D_n)$ where each $C_i (i \in \{1, \ldots, n\})$ is a predicate called a guard condition that contains no output variable and each $D_i (i \in \{1, \ldots, n\})$ is another predicate called defining condition that defines the output variables.

```
Input = set of nat = [W1,H1,D1]
Output = set of nat = [W2,H2,D2]
Hyperparameter = set of nat = [K,F,S,P]

process ShapeVerify(Input,Output,Hyperparameter:
    set) y:bool
pre forall[x:set] | x>0
post W2=(W1-F+2P)/S+1 and H2=(H1-F+2P)/S+1 and D2=
    K
    and y = true
    or
    W2<>(W1-F+2P)/S+1 or H2<>(H1-F+2P)/S+1 or D2<>
    K
    and y = false
end_process
```

Listing. 1. A formal specification of neural networks using SOFL.

Listing 1 shows a formal specification for convolutional neural network code using SOFL. *ShapeVerify* is a process that used to verify the dimensional relationship between tensors in neural network. It has pre-condition $S_{pre} := true$ and post-condition $S_{post} := ((W2 = (W1 - F + 2P)/S + 1) \wedge (H2 = (H1 - F + 2P)/S + 1) \wedge D2 = K \wedge y = true) \vee ((W2 <> (W1 - F + 2P)/S + 1) \vee (H2 <> (H1 - F + 2P)/S + 1) \vee (D2 <> K) \wedge y = false)$. Then, we can write the post-condition in the form of Definition 3:

$G_1 := ((W2 = (W1 - F + 2P)/S + 1) \wedge (H2 = (H1 - F + 2P)/S + 1) \wedge D2 = K$

$D_1 := y = true$

$G_2 := ((W2 <> (W1 - F + 2P)/S + 1) \vee (H2 <> (H1 - F + 2P)/S + 1) \vee (D2 <> K))$

$D_1 := y = false$.

According to the above definition, we will collect the specifications of different neural network models and express them using SOFL. Finally, the specification described in SOFL is accurately transformed into Python code to remove the test cases that do not meet the specification. The details about the syntax of SOFL can be found in the publication by Liu *et al.* [10].

III. PROBLEM FORMULATION AND OVERVIEW

In this chapter, we first formulate the problem [11] and the proposed method. Then, an overview of TBEM is given.

A. FORMALIZATION OF PROBLEM

In order to more clearly describe the problem that we solve and the methods to be proposed. We formalize the deep learning framework and the operation process of graph model in the framework.

- Formalization of model operation process in deep learning framework [12]. Let's define set API as

$$\mathbf{I} = \{A_i\}_{i=1}^{n} = \{A_1, A_2, \ldots, A_n\}$$

where A_i is the existing API in the neural network framework, and n is the number of $\mathbf{I}$. At the same time, The set of hyper-parameters to be set for each $\mathbf{I}$ is defined as

$$\mathbf{HP_{A_i}} = \{p_{A_i}^{j}\}_{j=1}^{n} = \{p_{A_i}^{1}, p_{A_i}^{2}, \ldots, p_{A_i}^{n}\}$$

where $p_{A_i}^{j}$ is the specific hyper-parameter to be set in each $\mathbf{I}$. A_i is the element in set API and j is the number of all Hyper-parameters of the $\mathbf{I}$. For example,

$$\sum_{i=1}^{n} |HP_{A_i}|$$

can represent the types of all settable hyper-parameters in the framework.

- Formal specification for deep learning model in framework [13]. Next, we describe the form of the model in the framework based on the definition of the deep learning framework. Given a set of $\mathbf{I}$, We do a finite Cartesian product

$$\overbrace{I \times I \times \cdots \times I}^{K}$$

denoted as I^K and

$$I^K = \{< A_1, A_2, \ldots, A_n > | A_i \in I, 1 \leq i \leq K\}$$

Then, the model in the deep learning framework can be defined as

$$model \in I^* = \bigcup_{K=1}^{\infty} I^K$$

B. FORMALIZATION OF METHOD

Because in the DL framework, the model usually runs in the form of calculation graph, so we mark the calculation graph [14] set as G. Let $CG : model \rightarrow G$ represents the mapping between the model and the calculation graph, for a given input $m \in model$, there will be a corresponding calculation garaph $g \in G$.

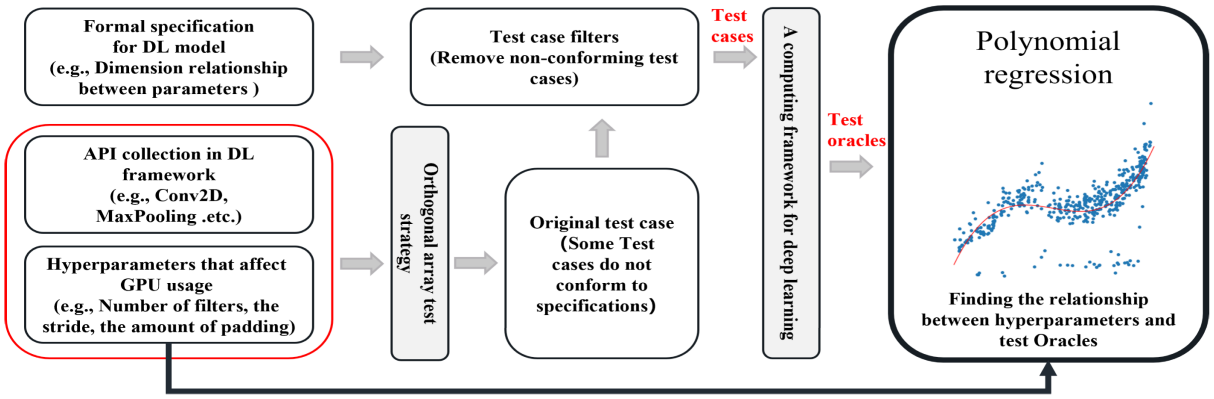

FIGURE 1. Overview.

Meanwhile, let GU be the set of interger means the size of the $GPU - memory$ consumed by the model, for each $m \in model$, a corresponding $GPU - memory$ usage can be obtained by running the model or static analysis for the graph [15]. Let $GV : model \rightarrow GU$ be a function of GU for the model. Through the investigation of previous studies, it can be seen that the static analysis of the calculation graph can roughly estimate the usage of the $GPU - memory$ of the model at run time. Therefore, let $SGV : graph \rightarrow GU$ be a function of GU for the model. There will be a certain gap between the GU obtained by static analysis of the model and the GU obtained by running the model. This gap can be defined as

$$Gap(model) = GV(model) - SGV(CG(model))$$

It is critical to define the relationship between $Gap(model)$ and $model$. Not only can it be used to get a more accurate model $GPU - memory$ usage, but it can also be used to evaluate the execution efficiency of the DL framework [16].

In order to find the specific mathematical form of $Gap(model)$, we propose a data fitting method based on OAGM. First, a certain scale of deep learning model is generated through the orthogonal array test strategy, which is used as a test case to test the GU value (Test Oracle) of the DL framework at runtime. Next, use the regression algorithm to find the relationship between test case and test oracle, that is, to obtain $GV(model)$. However, if we want to know $Gap(model)$, we still need to know the specific value of SGV. This paper adopts the method of static analysis of the computational graph, and evaluates the specific value of $SGV(CG(model))$ [2]through the analysis of the tensor scale.

C. OVERVIEW AND CHALLENGES

Figure 1 shows an overview of how TBEM works. The process is divided into test phase and data analysis phase. The test phase includes the generation of test cases and the collection of test data. In the data analysis phase, polynomial regression is used to solve the relationship between different hyperparameters of the neural network and GPU usage.

In the test phase, we need to automatically generate test cases. We believe that there are two principles for the generated test cases: (1) The number of generated test cases can be executed in a limited time. For example, suppose we select 10 Super parameters of neural network, and each super parameter has 3 values. The full test of such a neural network model will produce 59049 (3^{10}) test cases. Such a test scale may not be completed in a limited time. (2) Test cases need to be evenly distributed in the test space as much as possible, which can make the conclusion of polynomial regression more accurate.

```
from tensorflow import keras
from tensorflow.keras import layers, models, Input
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Conv2D,
    MaxPooling2D, Dense, Flatten, Dropout

def VGG16(classes, input_shape):
    input = Input(shape=input_shape)
    # 1st block
    x = Conv2D(64, 3, activation='relu')(input)
    x = Conv2D(64, 3, activation='relu')(x)
    x = MaxPooling2D(2, strides=(2,2))(x)
    # 2nd block
    x = Conv2D(128, 3, activation='relu')(x)
    x = Conv2D(128, 3, activation='relu')(x)
    x = MaxPooling2D(2, strides=(2,2))(x)
    # 3rd block
    ...
    # full connection
    x = Flatten()(x)
    x = Dense(4096,activation='relu')(x)
    x = Dense(4096, activation='relu')(x)
    output_tensor = Dense(classes, activation='
    softmax')(x)

    model = Model(input, output)
    return model
```

Listing. 2. VGG 16 model.

In order to overcome the above difficulties, we propose a test case generation algorithm based on OAGM. Firstly, the structure of many classical neural network models is defined by string formatting. Then, the OAGM strategy is used to deform the layers of the classical neural network template to produce a sufficient number of neural network structures. Finally, using OAGM again, the super parameters corresponding to the neural network structure are generated,

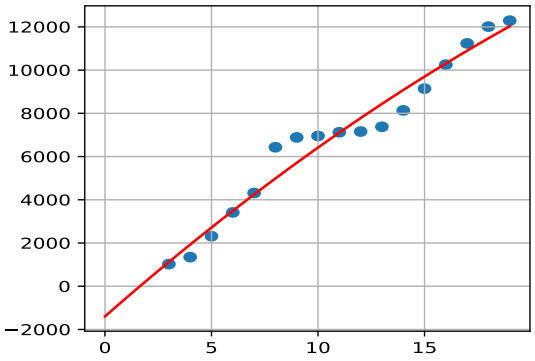

FIGURE 2. Relationship between the number of neural network layers and GPU utilization.

and the initial test cases are obtained.However, the test cases generated in this way cannot guarantee that they all meet the requirements of neural network framework. The main problem is that the shape of tensor may be inconsistent. Therefore, we implement a filter to remove the non-conforming test cases, so as to get the final test cases that can be run.

After the test is complete, we can get the corresponding GPU usage for different test cases. Then, how to use the data and solve the mapping relationship between test cases and GPU usage is an important issue. We propose a polynomial regression solution, which abstracts test cases into a hyperparametric vector and establishes a mapping relationship between the hyperparametric vector and the GPU usage. This process enables TBEM to infer the GPU usage of different neural network models. Finally, we make an empirical study on the reasoning ability of TBEM, which proves the validity of TBEM.

D. CASE STUDY

To clarify the role of regression in this algorithm, we give an example of univariate polynomial regression in this subsection. From the method of static analysis, the number of layers of deep learning is directly related to the consumption of computing resources. The function of univariate polynomial regression is to analyze the tested data and get the mathematical expression of the relationship between the number of deep learning layers and the computing resources.Finally, it is worthwhile to estimate the display memory consumption of various neural networks by using the regression results.

Visual Geometry Group(VGG) [17] is a classical neural network model. Listing 2 shows the implementation of the VGG16 model in the TensorFlow framework. In order to test the memory usage of training in the TensorFlow framework for in-depth learning models of different layers, and to ensure that the model is true and effective as possible.We mutated the VGG16 model.Because case study is the reason, the convolution layer parameters (filters, kernel size, padding, strides operations, etc.) and pool layer parameters (pool size, strides, padding, data format, etc.) appearing in the model are set as uniform parameters when mutating VGG16, excluding the effect of Hyperparameters other than the number of layers on explicit memory consumption in the deep learning model above. Figure 2 shows the relationship between the number of neural network layers and GPU usage (MB).

Algorithm 1: Automatic Generation of DL Test Model

Input: *hp* : Hyperparameters in neural networks
hp_list : A set of Hyperparameter
template : Verified valid DL mode
seed : A set of template
specification : Filter nonconfirming models
Output: *source_code*

```
parameterMatrix = Orthogonal(hp_list)
model = mutation(seed)
for each p in parameterMatrix do
    sourceCode_list = toString(model, p);
    for each sourceCode in sourceCode_list do
        if specification(sourceCode) is False then
            sourceCode_list =
              sourceCode_list.pop(sourceCode)
        end
    end
end
return sourceCode_list;
```

IV. APPROACH

In this section, we first introduce what OAGM is and how to use OAGM to generate test cases that can test the DL framework, and then analyze the feasibility of test cases generated based on OAGM and the ability of the generated test cases and test oracle to be applied to regression analysis feasibility. Finally, the regression model and static analysis model used in this method are introduced.

A. TEST CASE GENERATION BASED OAGM

The purpose of testing is to find out how much $GPU - memory$ is consumed by different models running under a certain framework. But there are many hyperparameters e.g., batch size [18].in the deep learning algorithm, and not all hyperparameter changes will have a huge impact on the $GPU - memory$. The static analysis of the deep learning calculation graph can filter out the APIs that have a greater impact on the memory consumption.

Observatios and motivations about API screening: GPU's advantage lies in parallel computing. In the process of training deep learning models, there are a large number of operators that need to be calculated in parallel. For example, feature mapping in forward propagation, gradient mapping in back propagation, etc. Therefore, we have screened APIs related to convolution operation, pool operation, and Batch Normalization that will generate a large number of parallel calculations. In each API, the input scale and output scale of each layer of neurons can be set, and different parameters correspond to different memory usage. In addition, the depth in deep learning is also a major

TABLE 1. Orthogonal test example of VGG network.

Test Number	Batch-size	Depth	Number of convolutional layers
Case1	4	11	8
Case2	8	13	10
Case3	16	16	13
Case4	32	19	16

factor in consuming memory. Therefore, in the test, models with different depths and different structures will be tested orthogonally [19]. Corresponding to the memory consumed by different models. Because the memory consumption of the underlying framework will not decrease with the increase of the influencing parameters in the model, that is, the direction of data change is known, and only the rate of change is unknown. Therefore, the data obtained by the orthogonal test is sufficient for multivariate polynomia regression analysis.

Let's take the VGG network as a example. Suppose that through the static analysis [20] of the neural network model, three representative hyperparameters are selected, namely Batch-size, Depth and Number of convolutional layers [21]. These three hyperparameters constitute the factors in the orthogonal array. As shown in Table 1. In an orthogonal array, the range of values for each factor is called levels. Table 1 shows an orthogonal array with a factor of 3 and levels of 4. If a comprehensive experimental method is used for testing, up to 3^4 tests are required. And the number of tests increases exponentially with the value of levels. However, using orthogonal experiments to generate orthogonal arrays requires only 4^2 tests. In other words, when the levels become very large, a comprehensive test is impossible. Therefore, this article uses the OAGM to generate test cases [22]. Testing the deep learning framework and record the test oracle corresponding to each test case.

B. POLYNOMIAL REGRESSION

we got a lot of pairs of test case and test Oracle, where test case is marked as *model* and test Oracle is marked as GU. Because the model is generated by transforming parameters. therefore, *model* can be denoted as $model(hyperparameter_1, \ldots, hyperparameter_n)$ and The hyperparameters are derived from the static analysis calculation graph.

Next, we use polynomial regression to find the relationship between hyperparameters and GU, which is equivalent to finding a way to solve $GV(model)$. Furthermore, $SGV(model)$ is known. We have also found a way to solve $Gap(model)$.

V. RELATED WORK

A. GPU-MEMORY ESTIMATION

So far, most of the research on memory management of DL accelerator-GPU focuses on how to optimize the use of GPU memory during model training. For example, Rhu *et al.* [23] proposed vDNN to formulate a memory swapping strategy between main memory and GPU memory by analyzing the computation graph, to reduce the footprint of GPU memory in the process of training. Gradient checkpoint [14] uses the idea of recomputation to implement an algorithm for training n layer network, which only consumes $\mathcal{O}(\sqrt{n})$ memory. SuperNeurons [15] and Capuchin [16] both combine memory sharing, memory swapping and recomputation techniques to varying degrees to further improve the optimization of GPU memory management in DL model training. However, unlike our work, these studies usually focus on how to optimize memory usage during DL model training. Rather than estimating how much GPU memory may be consumed by the model itself when the model is not trained.

DNNMem [4] is the most relevant work with TBEM. TBEM and DNNMem have a common purpose, that is, the GPU memory that may be consumed by the DL model is estimated before the DL model is executed. However, the TBEM method is based on the regression model generated from test data to evaluate the DL model. It is essentially different from DNNMem, which evaluates the GPU memory consumption of the DL model through the static analysis of the computation graph. Different working principles make TBEM overcome the following problems of static analysis methods in the following aspects: 1) the version update speed of each component of deep learning framework is fast, and the update cost of tools developed based on static analysis principle is large; 2) The dependency of deep learning framework is complex, and some components cannot be completely statically analyzed.

B. TESTING OF DL FRAMEWORK

In recent years, with the increasing demand for the stability of DL framework, the research on automatic test DL framework has gradually attracted the attention of researchers. Cradle [6] detects the inconsistency between the implementation of the same neural network model in multiple DL frameworks, determines that there may be errors in the inconsistent framework by using the way that the minority obeys the majority, and puts forward the relevant algorithm to locate the wrong location. Gao *et al.* [5] Proposed another DL framework testing method called Audee, which is different from cradle's practice of using existing DNNS as test cases. Audee tried to generate test cases by using search algorithm, and improved the bug type detection range and bug location accuracy.

For algorithms without multiple implementations, that is, when cross referencing cannot be used for Test Oracle comparison, Murphy *et al.* [24], [25] Tried to test the machine learning framework with the method of geometric relations, but the framework here is not a neural network framework.

The above methods mainly focus on how to detect the bugs in the framework. Although TBEM also needs to generate test cases for the framework to run, on the contrary, TBEM is correct based on the framework. We record the feedback given by the framework to prevent GPU memory overflow due to hyperparameter setting errors in neural network code implementation.

VI. CONCLUSION AND FUTURE WORK

In this article, we propose a test-based method to evaluate the memory usage of the DL framework. This method is different from the previous static analysis method. The possible errors of the static analysis method can be corrected through testing, and the possible GPU-memory usage of the deep learning model can be better evaluated before the deep learning model is running.

At present, it is only a theoretical framework. In the future, this method will be used to automatically generate a large number of test cases to test the mainstream DL framework, so as to prove the effectiveness of this method [26].

REFERENCES

[1] L. Floridi and M. Chiriatti, "GPT-3: Its nature, scope, limits, and consequences," *Minds Mach.*, vol. 30, no. 4, pp. 681–694, Nov. 2020.
[2] R. Zhang, W. Xiao, H. Zhang, Y. Liu, H. Lin, and M. Yang, "An empirical study on program failures of deep learning jobs," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, Jun. 2020, pp. 1159–1170.
[3] N. Humbatova, G. Jahangirova, G. Bavota, V. Riccio, A. Stocco, and P. Tonella, "Taxonomy of real faults in deep learning systems," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, Jun. 2020, pp. 1110–1121.
[4] Y. Gao, Y. Liu, H. Zhang, Z. Li, Y. Zhu, H. Lin, and M. Yang, "Estimating GPU memory consumption of deep learning models," in *Proc. 28th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Nov. 2020, pp. 1342–1352.
[5] Q. Guo, X. Xie, Y. Li, X. Zhang, Y. Liu, X. Li, and C. Shen, "Audee: Automated testing for deep learning frameworks," in *Proc. 35th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Sep. 2020, pp. 486–498.
[6] H. V. Pham, T. Lutellier, W. Qi, and L. Tan, "CRADLE: Cross-backend validation to detect and localize bugs in deep learning libraries," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng. (ICSE)*, May 2019, pp. 1027–1038.
[7] L. Lazić, "Use of orthogonal arrays and design of experiments via Taguchi methods in software testing," *Recent Adv. Appl. Theor. Math.*, pp. 256–67, 2013.
[8] H. Wu, "Application of orthogonal experimental design for the automatic software testing," in *Applied Mechanics and Materials*, vol. 347. Bäch, Switzerland: Trans Tech Publications, 2013, pp. 812–818.
[9] M. Vesely, "Computer curve fitting of polynomials," Illinois Univ. Urbana Coordinated Science Lab, Urbana, IL, USA, Tech. Rep. AD0758312, 1972.
[10] S. Liu, A. J. Offutt, C. Ho-Stuart, Y. Sun, and M. Ohba, "SOFL: A formal engineering methodology for industrial applications," *IEEE Trans. Softw. Eng.*, vol. 24, no. 1, pp. 24–45, Jan. 1998.
[11] S. Liu and S. Nakajima, "Automatic test case and test Oracle generation based on functional scenarios in formal specifications for conformance testing," *IEEE Trans. Softw. Eng.*, vol. 48, no. 2, pp. 691–712, Feb. 2022.
[12] S. A. Seshia, A. Desai, T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, S. Shivakumar, M. Vazquez-Chanlatte, and X. Yue, "Formal specification for deep neural networks," in *Proc. Int. Symp. Automated Technol. Verification Anal.* Cham, Switzerland, Springer, 2018, pp. 20–34.
[13] T. Dreossi, S. Ghosh, A. Sangiovanni-Vincentelli, and S. A. Seshia, "A formalization of robustness for deep neural networks," 2019, *arXiv:1903.10033*.
[14] T. Chen, B. Xu, C. Zhang, and C. Guestrin, "Training deep nets with sublinear memory cost," 2016, *arXiv:1604.06174*.
[15] L. Wang, J. Ye, Y. Zhao, W. Wu, A. Li, S. L. Song, Z. Xu, and T. Kraska, "Superneurons: Dynamic GPU memory management for training deep neural networks," in *Proc. 23rd ACM SIGPLAN Symp. Princ. Pract. Parallel Program.*, 2018, pp. 41–53.
[16] X. Peng, X. Shi, H. Dai, H. Jin, W. Ma, Q. Xiong, F. Yang, and X. Qian, "Capuchin: Tensor-based GPU memory management for deep learning," in *Proc. 25th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2020, pp. 891–905.
[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
[18] M. Papini, M. Pirotta, and M. Restelli, "Adaptive batch size for safe policy gradients," in *Advances in Neural Information Processing Systems*, vol. 30. Red Hook, NY, USA: Curran Associates, 2017, pp. 3594–360.
[19] S. Liu and S. Nakajima, "A 'vibration' method for automatically generating test cases based on formal specifications," in *Proc. 18th Asia–Pacific Softw. Eng. Conf.*, Dec. 2011, pp. 73–80.
[20] P. M. Radiuk, "Impact of training set batch size on the performance of convolutional neural networks for diverse datasets," *Inf. Technol. Manage. Sci.*, vol. 20, no. 1, pp. 20–24, Jan. 2017.
[21] S. Mittal and S. Vaishay, "A survey of techniques for optimizing deep learning on GPUs," *J. Syst. Archit.*, vol. 99, Oct. 2019, Art. no. 101635.
[22] B. Di, J. Sun, D. Li, H. Chen, and Z. Quan, "GMOD: A dynamic GPU memory overflow detector," in *Proc. 27th Int. Conf. Parallel Archit. Compilation Techn.*, 2018, pp. 1–13.
[23] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. W. Keckler, "VDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–13.
[24] C. Murphy, G. E. Kaiser, and M. Arias, "An approach to software testing of machine learning applications," Dept. Comput. Sci., Columbia Univ., New York, NY, USA, Tech. Rep. CUCS-014-07, 2007.
[25] C. Murphy, G. E. Kaiser, and L. Hu, "Properties of machine learning applications for use in metamorphic testing," Dept. Comput. Sci., Columbia Univ., New York, NY, USA, Tech. Rep. CUCS-011-08, 2008.
[26] H. Liu, S. Liu, and A. Liu, "Testing-based GPU-memory consumption estimation for deep learning," in *Proc. Softw. Eng. Symp.*, Aug. 2021, pp. 196–199.

HAIYI LIU received the M.Sc. degree in mathematics from the Guilin University of Electronic Technology, China. He is currently pursuing the Ph.D. degree in computer science with Hiroshima University, Japan. His research interests include testing and verification of deep learning AI systems and software stability.

SHAOYING LIU (Fellow, IEEE) received the B.Sc. and M.Sc. degrees in computer science from Xi'an Jiaotong University, China, and the Ph.D. degree in computer science from The University of Manchester, U.K. He is currently a Professor in software engineering at Hiroshima University. He has published one book, 12 edited books, and more than 250 papers in journals and conferences. His research interests include formal engineering methods, software testing, human–machine pair programming, and intelligent software engineering environments. He is a BCS Fellow.

CHENGLONG WEN received the B.Sc. degree in computer science from Nanyang Normal University, China. He currently works as a Software Development Engineer at the Microsoft Software Technology Center Asia. His research interests include software reliability and network security.

W. ERIC WONG (Senior Member, IEEE) received the M.S. and Ph.D. degrees in computer science from Purdue University, West Lafayette, IN, USA. He is currently a Full Professor, the Director of International Outreach, and the Founding Director of the Advanced Research Center for Software Testing and Quality Assurance in Computer Science, The University of Texas at Dallas (UTD). He also has an appointment as a Guest Researcher at the National Institute of Standards and Technology, an agency of the U.S. Department of Commerce. Prior to joining UTD, he was with Telcordia Technologies (formerly Bellcore) as a Senior Research Scientist and the Project Manager in charge of dependable telecom software development.