一种基于动态图与代码执行上下文感知的微服务异常检测方法

顾加伟[1,2]，姜瑛[1,2*]，杨天晴[1,2,3]
1. 昆明理工大学 云南省人工智能重点实验室，云南 昆明 650504；
2. 昆明理工大学 信息工程与自动化学院，云南 昆明 650504；
3. 保山学院 大数据学院，云南 保山 678000

摘要
针对微服务异常检测在方法级交互复杂、容器实例高度动态以及异常样本稀缺等约束，提出一种基于动态图与代码执行上下文感知的微服务异常检测方法DGCEC-MAD。通过插桩技术采集服务调用对应所运行的代码信息，包含其类名、方法名、参数与返回值等执行语义，经预训练句向量编码；将服务调用建模为带时间戳与边属性的连续时间动态图，利用可更新的节点内存与融合时间及边属性的图注意力机制刻画关系演化；采用改进的链接预测任务结合双重负采样机制从正常数据中学习调用模式，获得高质量节点表示后与KPI融合，最终通过自编码器重构误差实现无监督异常检测。实验在两套微服务系统及容器迁移等动态场景中验证了方法的有效性，消融实验进一步表明，引入代码执行上下文可精细刻画方法级行为，提升异常检测效果。

关键词
云计算；微服务；容器化；异常检测；动态图；自编码器

中图分类号：TP311　　文献标志码：A　　doi：10.11959/j.issn.2096-0271.2026039

A dynamic graph and code execution context-aware microservice anomaly detection method

GU Jiawei[1,2], JIANG Ying[1,2*], YANG Tianqing[1,2,3]
1. Yunnan Provincial Key Laboratory of Artificial Intelligence, Kunming University of Science and Technology, Kunming Yunnan 650504, China；
2. School of Information Engineering and Automation, Kunming University of Science and Technology, Kunming Yunnan 650504, China；
3. School of big data, Baoshan University, Baoshan, Yunnan 678000, China

Abstract
The heterogeneity of method-level interactions, the high dynamics of container instances, and the scarcity of anomaly labels make anomaly detection in microservice systems difficult. Therefore, a dynamic graph and code execution context-aware microservice anomaly detection method (DGCEC-MAD) was proposed. Firstly, code information corresponding to service calls was collected through instrumentation techniques, including execution semantics such as class names, method names, parameters, and return values, which were encoded using a pre-trained sentence

embedding model. Secondly, service calls were modeled as a continuous-time dynamic graph with timestamps and edge attributes; updatable node memories and a graph attention mechanism that fuses temporal and edge features were employed to capture the evolution of inter-service relations. Then, an improved link prediction task with dual negative sampling was designed to learn normal calling patterns purely from normal data and obtain high quality node representations. Finally, the learned representations were fused with KPIs, and anomalies were identified in an unsupervised manner via autoencoder reconstruction errors. Experiments on two microservice systems, including container-migration scenarios, demonstrate superior performance and robustness, with consistent gains over strong dynamic-graph baselines. Ablation studies further show that incorporating code execution context finely characterizes method-level behavior and significantly improves detection effectiveness.

Key words
cloud computing, microservices, containerization, anomaly detection, dynamic graph, autoencode

0 引言

随着大数据技术的快速发展和数据量的爆炸式增长[1]，传统单体架构因在部署、扩展和维护上的局限性，已难以满足高并发、大规模业务需求。在此背景下，微服务架构[2]应运而生，其核心思想在于将单体应用拆解为多个功能单一、松耦合且可独立部署的细粒度服务单元，并借助容器化技术[3]实现服务的轻量级封装与跨平台部署，显著提升了系统的敏捷性和可维护性。然而，微服务架构的分布式特性使得服务间存在复杂的调用依赖关系，单个服务的异常往往通过调用链传播，引发连锁故障，甚至导致整个系统瘫痪。因此，准确的异常检测成为保障微服务系统稳定运行的关键技术。传统异常检测方法在微服务环境下效果不佳，本文分析认为三个关键因素直接影响检测效果：

1）方法级调用的异质性导致正常与异常模式边界模糊：现有研究多将多样化的方法级调用统一抽象为服务间调用[4-6]，仅通过调用开销、频次等统计特征进行建模，忽略了方法本身在业务逻辑和语义上的差异。这种粗粒度的统计表征难以捕捉由方法调用模式差异所引发的异常，导致部分异常信号被掩盖。实际上，微服务系统中常常存在性能指标相近但业务语义完全不同的方法调用，因此有必要引入细粒度的调用属性和方法级语义上下文，以增强异常检测的区分能力。

2）容器实例的高度动态性破坏检测模型的分布稳定性：容器实例并非静态绑定于特定计算节点，容器编排平台（如Kubernetes）会根据集群资源利用率、网络拓扑等多维度因素，动态执行容器的创建、销毁、迁移和扩缩容操作，使得拓扑与负载持续变化，传统静态拓扑方法难以适配，导致训练与真实场景分布不匹配。

3）异常样本的稀缺性限制检测模型的学习能力：微服务系统在正常运行中积累的大量调用数据蕴含着丰富的运行规律和交互模式。传统监督方法依赖稀缺的异常样本，现有无监督方法则缺乏对正常数据内在规律的深度挖掘，基于正常模式学习实现精准异常检测成为难点。

为此，本文提出基于动态图与代码执行上下文感知的微服务异常检测方法DGCEC- MAD（Dynamic Graph and Code Execution Context Aware

Microservices Anomaly Detection）。通过插桩采集代码执行上下文并利用预训练语言模型生成语义向量，将服务调用建模为带时间戳的动态图事件流，采用改进的链接预测结合自编码器的无监督框架从正常数据中学习调用规律。实验表明，DGCEC-MAD在多组数据上的检测结果有一定提升，引入代码执行上下文后能有效刻画方法层面的功能差异；在容器迁移等动态场景中仍保持稳定性能，体现出良好的鲁棒性与适应性。本研究为应对微服务系统中异常检测的多重挑战提供了一种综合且有效的解决方案。

1 相关工作

1.1 异常检测

传统异常检测方法主要包括统计分析和聚类技术，如单类支持向量机（One-Class SVM）[9]、局部离群因子（LOF）[10]、K-means[11]聚类和孤立森林[12]等。这些方法通过构建决策边界、分析密度差异或聚类分布来识别异常点，适用于较为稳定的数据环境。然而在云环境下，由于依赖固定统计特征和预设参数，这类方法难以应对数据分布的动态变化，检测效果受到一定限制。

为克服传统方法的局限，近些年研究者纷纷引入深度学习技术，以自动学习数据中的非线性和复杂时序特征。譬如，TraceAnomaly[5]方法通过提取微服务调用路径与响应时间数据构建跟踪向量，再利用深度贝叶斯网络或核密度估计方法对正常模式进行建模，从而实现异常检测；SLA-VAE[13]结合变分自编码器与关键性能指标（Key Performance Indicator，KPI）及残差误差对异常进行判定。这些方法往往忽略对服务实例间实际物理或逻辑拓扑依赖信息的利用，在面对具有局部依赖性的异常时仍难以获得理想效果。

考虑到微服务间有着复杂的依赖关系，基于图神经网络（Graph Neural Network，GNN）[14]的异常检测方法受到越来越多的关注。这类方法将服务容器视为节点，利用调用链、性能指标等数据构建图结构，并采用图模型进行特征提取。例如，TopoMAD[7]提出GraphLSTM架构，通过将传统LSTM[15]中的全连接层替换为GNN层，使得每个节点在更新隐藏状态时能够同时考虑自身的时序信息和邻居节点的空间信息；TraceGra[4]构建跟踪性能图，将调用跟踪信息与容器性能指标融合，同时通过LSTM编解码器和GCN[16]编解码器来学习正常的时空模式；DCADAnomaly[17]采用双通道GIN，分别从大量正常与少量异常样本学习图级表示，首先初始化各通道中心，再分别训练通道，最终以图嵌入到对应中心的距离作为异常分数。尽管这些方法在建模拓扑时空变化方面取得了一定进展，但仍存在两个关键局限：一方面，通常基于固定的邻接矩阵构建图结构，难以适应云环境中容器实例动态创建、销毁、迁移等导致的拓扑结构变化；另一方面，主要依赖性能指标或调用统计信息（如响应时间、调用频次）等数值特征，将复杂多样的方法调用统一抽象为简单的连接关系，忽略了不同调用在功能语义层面的本质差异。

1.2 动态图

与静态图（节点与边保持不变）不同，动态图[18]指拓扑与属性随时间演化的图，如何在图中对动态过程进行建模是其研究重点。DyRep[19]通过时间注意力与递归式

状态更新将演化规律映射到低维节点表示。TGAT[20]基于自注意力提出函数式时间编码，在历史时间邻域上进行时间感知的特征聚合。TGN[21]为每个节点设计可更新的内存状态用来捕获时间信息，再经GNN生成节点表示，兼顾连续时间流式训练、长程依赖。GraphMixer[22]基于多层感知机(MLP)设计特征时间混合器汇总时序链路信息，通过邻居均值池化来捕捉节点的属性特征，在训练过程中表现出更快收敛性。

综上，现有异常检测方法在调用建模、动态拓扑适应存在一定不足。部分研究已尝试采用无监督的图模型方法如TopoMAD、TraceGra，但仅基于简单的统计特征或静态结构，缺乏对正常调用数据中时空演化规律和复杂交互模式的深度挖掘。而动态图具备连续时间建模、长期记忆与归纳能力，更适合于微服务的动态场景。因此，本文提出了DGCEC-MAD方法针对性地解决上述问题。通过插桩采集服务调用的执行语义信息，并利用预训练语言模型进行高质量语义编码，将语义编码与调用链属性共同构成边事件流以作为TGN动态图的输入，从而摆脱使用静态的邻接矩阵，实现对服务调用关系及其动态演化的精准建模。以改进的链接预测任务驱动动态图优化，深度挖掘正常调用的时空规律，获得高质量节点表征，异常检测阶段将节点表征与KPI融合，利用自编码器重构误差实现无监督检测。其充分利用系统正常运行中积累的丰富调用数据，有效提升了对复杂动态环境下异常的检测准确性，克服了现有方法在语义建模和动态拓扑适应方面的不足。

2 基于动态图与代码执行上下文感知的微服务异常检测方法

为实现异常检测，DGCEC-MAD主要包括数据收集、图构建、动态图学习以及异常检测四个模块，整体框架见图1。方法以插桩采集的执行上下文、调用链属性与KPI为基础，首先将服务调用构造成带时间戳与边属性的连续时间动态图，随后通过时间编码、内存更新与融合时间与边特征的图注意力学习节点表示，最后在节点表示与KPI的后期融合基础上，利用自编码器的重构误差实现无监督异常判定。

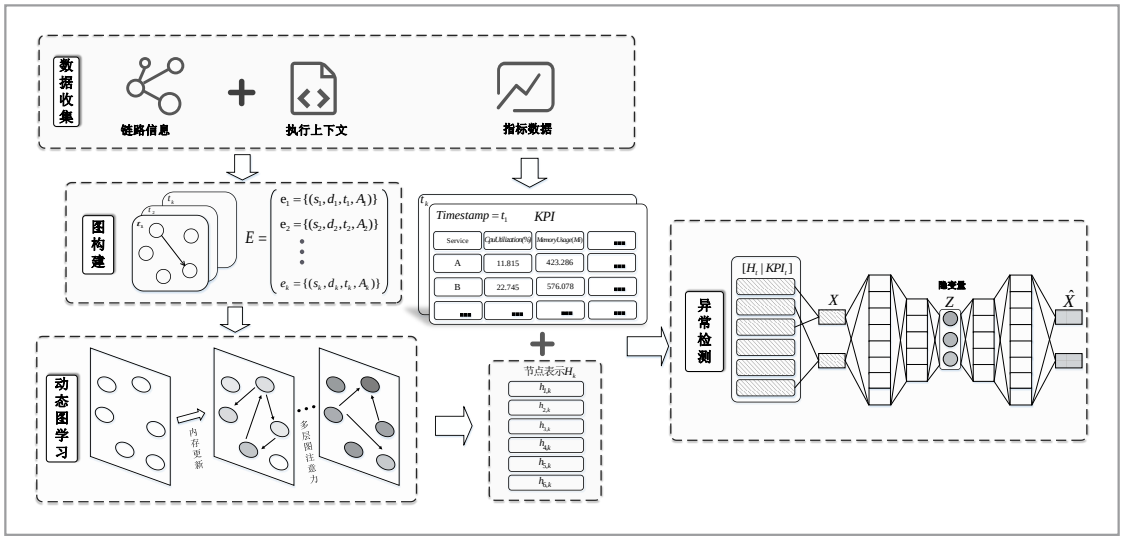

图1 DGCEC-MAD框架

2.1 数据收集

本文收集了微服务系统运行过程中的多种数据，包括KPI、调用链信息以及服务调用的执行上下文信息。KPI分为基础设施层面的节点指标和应用层面的容器指标，涵盖CPU利用率、内存使用率、磁盘I/O性能、网络吞吐量等。数据采集完成后，对KPI等数值型数据进行归一化处理。

分布式调用链通过唯一的Trace ID标识，包含多个跨度（Span）。如图2左侧所示，每个Span代表一次具体的服务调用，Preserve Service作为某次调用的根Span通过跨调用服务形成层次化的调用树结构。

然而，如引言中所述，仅依据Span层级的统计信息（如持续时间、调用频次）不足以识别由调用模式差异引发的异常。由于具体业务逻辑实质是由Span内部的方法级代码执行产生的，为解决方法级调用的异质性问题，本文设计了基于代码执行上下文的细粒度语义刻画方案。如图2右侧所示，在微服务应用架构的三个关键层次部署插桩代码：控制层（Controller，负责处理HTTP请求）、服务实现层（ServiceImpl，实现具体业务逻辑）以及服务层（Service，执行底层服务调用）。插桩代码在这三层的方法执行过程中实时记录类名、方法名、参数类型及返回值等信息，从而使每个Span关联到相应的代码执行上下文。

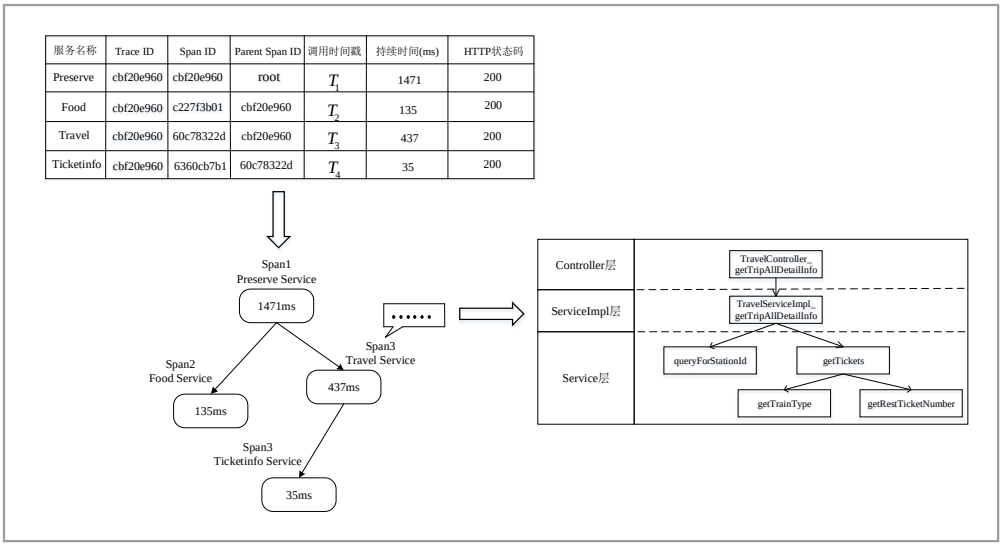

图2 分布式调用链与代码插桩示例

在执行上下文的编码方法选择上，尽管大语言模型（Large Language Model，LLM）具备较强的语义建模能力，但其推理成本较高。鉴于本任务上下文多为结构化的短文本，轻量级句向量模型已足以满足表征需求。基于此，本文采用预训练句向量模型paraphrase-MiniLM-L6-v2[23]进行编码。该模型构建于Sentence-BERT[24]框架之上，并在大规模高质量语料上充分预训练，能够直接对标准化的上下文文本进行高效且准确的语义表示。在此基础上，为降低计算开销和过拟合风险，

进一步对其固定生成的384维语义向量采用主成分分析（Principal Component Analysis，PCA）进行降维，选择累计方差贡献率达到95%的主成分，将特征维度压缩至12维，从而在保留主要语义信息的同时提升训练效率与稳定性。

2.2 图构建

在数据预处理的基础上，本文将微服务系统建模为连续时间动态图 $G=(V,E)$。其中，V为节点集合，每个节点对应于调用链中的一个具体Span，即某次服务调用实际运行时所关联的容器实例。E为边事件流，是由所有按时间顺序发生的服务间调用事件组成的有序集合。每条边事件 $e=(s,d,t,A)$表示源节点s在时间t向目标节点d发起的有向通信，属性A由预处理阶段生成的特征向量构成，具体包括调用链数据（如服务名称、调用时间、持续时间、HTTP状态码等）以及调用的执行上下文语义特征。将一系列信息纳入边属性特征A是为避免云环境下的容器动态迁移会导致节点属性不稳定的情形。

随着系统运行，链路数据不断记录服务调用行为，每一次实际发生的服务调用都被抽象为一个边事件，所有边事件按时间顺序组成边事件流，从而动态更新图结构。

2.3 动态图学习

2.3.1 时间编码

考虑到微服务系统中事件流的连续性和计算效率的需求，根据公式（1-2）采用时间批次处理策略，将连续的时间流划分为固定长度的时间窗口：

$$\mathcal{T}_{\text{batch}}=[t_{\text{start}},t_{\text{end}}] \tag{1}$$

$$\mathcal{E}_{\text{batch}}=\{(s_k,d_k,t_k,A_k):t_k\in\mathcal{T}_{\text{batch}}\} \tag{2}$$

其中，(s_k,d_k,t_k,A_k)表示在时间t_k发生的从节点s_k到节点d_k的交互事件，A_k为边属性特征向量。为有效建模节点间交互的时序特性，对批次内的每次交互事件，计算时间间隔，如公式（3）所示：

$$\Delta t_{sd}(t)=t-t_{sd}^{\text{last}} \tag{3}$$

其中，t_{sd}^{last}表示节点s和d上次交互的时间。随后通过时间编码器将时间间隔映射为高维向量表示，时间编码采用余弦函数实现，如公式（4）所示：

$$\tau_{sd}(t)=\text{TimeEncoder}(\Delta t_{sd}(t))=\cos(\mathbf{w}\cdot\Delta t_{sd}(t)+\mathbf{b}) \tag{4}$$

其中，$\mathbf{w}\in\mathbb{R}^{d_\tau}$是可学习的权重向量，$\mathbf{b}\in\mathbb{R}^{d_\tau}$为偏置向量，$d_\tau$为时间编码维度。时间编码$\tau_{sd}(t)$能够有效表征交互的时序特征，为后续的消息构建提供丰富的时序信息。

2.3.2 内存更新

当节点s和节点d之间发生交互边事件e时，通过公式（5）构建消息向量：

$$m_s(t)=\text{MLP}(\text{Concat}(s_s(t^-),s_d(t^-),\tau_{sd}(t),A_{sd}(t))) \tag{5}$$

其中，Concat是拼接操作，将节点s和d在时间t^-的历史状态、时间编码$\tau_{sd}(t)$以及属性$A_{sd}(t)$融合。为应对同一批次中涉及相同节点存在多个事件消息情况，使用聚合机制agg以缩短消息处理的开销，如公式（6）所示：

$$\tilde{m}_s(\mathcal{T}_{\text{batch}})=\text{agg}(\{m_s(t_k):t_k\in\mathcal{T}_{\text{batch}},s\in\text{event}(t_k)\}) \tag{6}$$

基于聚合后的消息，采用门控循环单元[25]更新节点的动态内存状态，如公式（7）所示：

$$s_s(\mathcal{T}_{\text{batch}}) = \text{GRU}(\tilde{m}_s(\mathcal{T}_{\text{batch}}), s_s(\mathcal{T}_{\text{batch}}^-)) \quad (7)$$

每个节点维护一个动态内存状态$s_s(t)$，其用于存储节点的历史交互信息。每当涉及节点s的事件发生时，模型会触发相应的内存状态更新。

2.3.3 图注意力

注意力机制已在特征融合与表示学习任务中得到广泛应用[26]。本文采用多层图注意力网络，基于节点的动态内存状态学习高阶表示。首先，将节点内存状态作为图神经网络的初始节点特征即$h_s^{(0)} = s_s(\mathcal{T}_{\text{batch}})$。与传统的GAT[27]模型仅考虑节点间注意力不同，将边的属性信息A和时间编码τ纳入注意力计算过程。对于第l层的节点表示学习，查询（Query）、键（Key）、值（Value）向量的构建如公式（8-10）所示：

$$q_{c,s}^{(l)} = \mathbf{W}_q^{(l)} h_s^{(l-1)} \quad (8)$$

$$k_{c,d}^{(l)} = \mathbf{W}_k^{(l)} \text{Concat}(h_d^{(l-1)}, A_{sd}, \tau_{sd}) \quad (9)$$

$$v_{c,d}^{(l)} = \mathbf{W}_v^{(l)} \text{Concat}(h_d^{(l-1)}, A_{sd}, \tau_{sd}) \quad (10)$$

其中，c表示注意力头的索引，$\mathbf{W}_q^{(l)}$、$\mathbf{W}_k^{(l)}$、$\mathbf{W}_v^{(l)}$分别为第l层的查询、键、值投影矩阵。基于构建的查询和键向量，计算注意力权重，如公式（11）所示：

$$\alpha_{c,sd}^{(l)} = \frac{\exp(q_{c,s}^{(l)\top} k_{c,d}^{(l)} / \sqrt{d})}{\sum_{u \in \mathcal{N}(s)} \exp(q_{c,s}^{(l)\top} k_{c,u}^{(l)} / \sqrt{d})} \quad (11)$$

其中，d为特征维度，$\mathcal{N}(s)$表示节点s的邻居集合。单头注意力的输出通过加权聚合邻居信息获得，如公式（12）所示：

$$\text{head}_{c,s}^{(l)} = \sum_{d \in \mathcal{N}(s)} \alpha_{c,sd}^{(l)} v_{c,d}^{(l)} \quad (12)$$

多头注意力机制通过从多个不同的“视角”对节点表示进行聚合，如公式（13）所示：

$$\text{MultiHead}_s^{(l)} = \text{Concat}_{c=1}^{C}(\text{head}_{c,s}^{(l)}) \cdot \mathbf{W}_o^{(l)} \quad (13)$$

其中，C为注意力头数，$\mathbf{W}_o^{(l)}$为输出投影矩阵。为了稳定训练过程并保留重要信息，添加残差连接[28]和层归一化[29]，如公式（14）所示：

$$h_s^{(l)} = \text{LayerNorm}(\text{MultiHead}_s^{(l)} + h_s^{(l-1)}) \quad (14)$$

经过L层图注意力网络的处理，获得最终的节点表示：$h_s = h_s^{(L)}$。

2.3.4 动态图损失函数

链接预测作为图神经网络的经典自监督学习任务，能够有效学习图的演化规律和节点间的潜在关系模式。然而，传统的链接预测方法在微服务异常检测场景中存在局限性，即仅依赖节点表示来判断边的存在性，如公式（15）所示：

$$\hat{y}_{sd} = \sigma(\text{MLP}(\text{Concat}(h_s, h_d))) \quad (15)$$

实际上，服务间的调用不仅取决于服务本身的特性，还与具体的调用边的属性特征密切相关。因此，为使学习到的节点表示更好地适配后续的异常检测任务，本文将属性特征直接纳入预测过程，如公式（16）所示：

$$\hat{y}_{sd} = \sigma(\text{MLP}(\text{Concat}(h_s, h_d, A_{sd}))) \quad (16)$$

此外，为进一步提升节点表示对异常模式的敏感性，本文设计了两种互补的负采样策略：1）随机节点采样：通过随机选择不存在的边作为负边，学习图的基本

结构模式，如公式（17）所示：

$$\text{NegSample}_1(s,d,t,A_{sd})=(s,d^-,t,A_{sd}) \quad (17)$$

其中，d^-表示随机选择不为目标节点d的负样本节点。2）属性特征扰动采样：考虑到某些异常调用其边事件属性特征中往往表现与正常调用模式有一定偏差，因而设计对边属性特征进行扰动的负采样，如公式（18）所示：

$$\text{NegSample}_2(s,d,t,A_{sd})=(s,d,t,\tilde{A}_{sd}) \quad (18)$$

其中，扰动后的边特征$\tilde{A}_{sd}=A_{sd}\cdot(1+\epsilon)$，$\epsilon\in\mathcal{N}(0,\theta^2)$，扰动强度$\theta\in[0.1,0.3]$。基于正负样本，链接预测的损失函数为二分类交叉熵损失，如公式（19）所示：

$$L_{\text{link}}=-\frac{1}{\left|\mathcal{E}_{\text{pos}}\right|+\left|\mathcal{E}_{\text{neg}}\right|}\sum_{(s,d)\in\mathcal{E}_{\text{pos}}\cup\mathcal{E}_{\text{neg}}}\left[y_{sd}\log\hat{y}_{sd}+(1-y_{sd})\log(1-\hat{y}_{sd})\right] \quad (19)$$

其中，$\mathcal{E}_{\text{pos}}$为正样本边集合，$\mathcal{E}_{\text{neg}}$为负样本边集合，$y_{\text{sd}}=1$表示正样本，$\widehat{y}_{sd}=0$表示负样本。

2.4 异常检测

2.4.1 输入构建

对于批次内的每条边事件(s_k,d_k,t_k,A_k)，首先构建源节点和目标节点的联合特征，如公式（20- 21）所示：

$$z_{src,k}=\text{Concat}(h_{s_k},\text{KPI}_{s_k}) \quad (20)$$

$$z_{dst,k}=\text{Concat}(h_{d_k},\text{KPI}_{d_k}) \quad (21)$$

其中，h_{s_k}和h_{d_k}分别为源节点s_k和目标节点d_k的图表示，KPI_{s_k}和KPI_{d_k}为对应节点的关键性能指标向量。随后，将源节点和目标节点的联合特征拼接，形成边级的综合特征表示，如公式（22-23）所示：

$$x_k=\text{Concat}(z_{src,k},z_{dst,k}) \quad (22)$$

$$\mathbf{X}=[x_1,x_2,...,x_{|\mathcal{E}_{\text{batch}}|}]^T\in\mathbb{R}^{|\mathcal{E}_{\text{batch}}|\times d_{input}} \quad (23)$$

其中，d_{input}为综合特征的维度。

2.4.2 样本重构

自编码器作为一种无监督学习模型，通过编码器将输入特征$\mathbf{X}$映射到潜在空间，解码器从潜在表示$\mathbf{Z}$重构原始特征，从而针对性学习正常模式的数据规律，如公式（24-25）所示：

$$\mathbf{Z}=\text{Encoder}(\mathbf{X})=\text{ReLU}(\mathbf{X}\mathbf{W}_{e1}+\mathbf{b}_{e1})\mathbf{W}_{e2}+\mathbf{b}_{e2} \quad (24)$$

$$\hat{\mathbf{X}}=\text{Decoder}(\mathbf{Z})=\text{ReLU}(\mathbf{Z}\mathbf{W}_{d1}+\mathbf{b}_{d1})\mathbf{W}_{d2}+\mathbf{b}_{d2} \quad (25)$$

其中，$\mathbf{W}_{e1}\in\mathbb{R}^{d_{input}\times d_{hidden}},\mathbf{W}_{e2}\in\mathbb{R}^{d_{hidden}\times d_{latent}}$为编码器权重矩阵，$\mathbf{W}_{d1}\in\mathbb{R}^{d_{latent}\times d_{hidden}},\mathbf{W}_{d2}\in\mathbb{R}^{d_{hidden}\times d_{input}}$为解码器权重矩阵，$\mathbf{b}_{e1},\mathbf{b}_{e2},\mathbf{b}_{d1},\mathbf{b}_{d2}$为对应的偏置向量，$d_{hidden}$为隐藏层维度，$d_{latent}$为潜在空间维度。

2.4.3 联合优化目标

对于每个样本x_k，计算其重构误差，如公式（26）所示：

$$R_k=\|x_k-\hat{x}_k\|^2=\sum_{j=1}^{d_{input}}(x_{k,j}-\hat{x}_{k,j})^2 \quad (26)$$

其中，$x_{k,j}$和$\hat{x}_{k,j}$分别为原始特征和重构特征的第j个维度。批次内的重构损失定义为所

有样本重构误差的均值，如公式（27）所示：

$$L_{\text{recon}}=\frac{1}{|\mathcal{E}_{\text{batch}}|}\sum_{k=1}^{|\mathcal{E}_{\text{batch}}|}R_k \tag{27}$$

训练阶段会联合优化动态图部分的链接预测损失和异常检测阶段的重构损失，如公式（28）所示：

$$L_{\text{total}}=\alpha L_{\text{link}}+\beta L_{\text{recon}} \tag{28}$$

其中，α和β为损失权重超参数，用于平衡两个学习目标的重要性。

2.4.4 异常判定

考虑到云环境的不断变化可能导致数据重构得分的整体波动，本文采用3σ准则，将阈值设为$\mu+3\sigma$，即以均值加三倍标准差作为判别标准，若重构分超过该阈值则判为异常，否则视为正常并存储至历史数据库。

3 实验结果与分析

3.1 数据集

为验证本文方法有效性与通用性，选取了开源基准系统TrainTicket微服务和自主开发包含13个功能模块（如订单服务、支付服务、消息推送等业务组件）的MallService微服务作为测试对象。

采用混沌工程在多层级注入故障：ChaosBlade用于节点/容器层的CPU、内存、丢包与网络延迟；Istio通过流量控制与延迟策略扰动服务间通信；Locust施加高并发与异常路径访问。上述组合覆盖节点、容器与网络，全面验证微服务鲁棒性。

针对两个微服务采用Locust负载测试工具分别模拟约10万次正常用户访问情形与1万次异常访问情形，结合Jaeger与Elasticsearch进行分布式链路追踪与持久化。使用Prometheus以3s间隔采集KPI，包括CPU使用率、内存占用、网络吞吐量等。此外，为评估动态环境适应性，在MallService原始数据上构建环境变更数据集：通过Kubernetes迁移约20%的服务实例至其他节点，模拟容器动态调度。

3.2 实验设置

时间编码维度为100；图注意力层数为2；注意力头数为2；损失权重α=0.3、β=0.7；批量大小为200；学习率为0.001；聚合方式使用平均聚合；按70%:15%:15%划分为训练、验证和测试集；使用早停策略进行训练。

在异常检测算法的评估中，采用广泛使用的精确率（Precision）、召回率（Recall）和F1分数作为性能指标。根据真实标签将检测结果分为真正例（TP，异常被判为异常）、假正例（FP，正常被误判为异常）和假负例（FN，异常被误判为正常），并分别计算Precision = TP / (TP + FP)，Recall = TP / (TP + FN)和F1 = 2 · Precision · Recall / (Precision + Recall)，以衡量算法的整体检测性能。

3.3 异常检测实验

本文选取OC-SVM、TraceAnomaly、TopoMAD、TraceGra、DCADAnomaly作为基线；同时对比DyRep、TGAT、GraphMixer三种主流动态图方法。为保证公平性，所有动态图方法在异常检测阶段均采用相同的自编码器结构，仅在动态图学习阶段的网络架构上存在差异。两个

微服务数据集的结果见表1。由于TrainTicket的服务数量更多、调用链更复杂，异常发生时受影响的服务范围更大，异常信号更显著，因此大多数方法在TrainTicket上的整体效果优于MallService。

表1 异常检测结果

方法	MallService			TrainTicket		
	Precision	Recall	F1	Precision	Recall	F1
OC-SVM	84.30	47.88	61.07	75.00	69.23	72.00
TopoMAD	85.76	68.58	76.21	85.15	76.67	80.70
TraceAnomaly	71.32	74.92	73.07	72.31	77.05	74.60
TraceGra	73.27	75.51	74.37	84.95	88.97	86.89
DCADAnomaly	82.14	62.16	70.77	85.37	81.40	83.33
DyRep+AE	81.64	92.03	86.52	87.72	95.58	91.48
TGAT+AE	86.95	87.63	87.29	89.00	96.73	92.70
GraphMixer+AE	85.80	91.00	88.32	87.76	**97.89**	92.55
DGCEC-MAD	**88.68**	**95.17**	**91.81**	**89.73**	96.75	**93.11**

实验表明，传统的OC-SVM以决策边界区分正常与异常，但微服务异常呈现多样性与动态性，涉及多维交互与时变依赖，已超出其单类假设的能力；TopoMAD虽引入静态交互信息，但难刻画频繁调度引起的拓扑变化；TraceAnomaly主要依赖响应时间，忽略资源与交互；TraceGra结合GNN与LSTM，但未充分利用节点与容器间的细粒度交互，系统状态表征仍不完整；DCADAnomaly利用大量未标记图和少量标记异常图进行训练，但在微服务环境下，异常类型多样，仅依赖局部的异常图难以全面覆盖所有异常模式，导致其Recall偏低。相比之下，显式建模拓扑演化的动态图方法整体更优，且在第二个数据集上优势更明显：以TraceAnomaly为参照，其F1在MallService为73.07、在TrainTicket为74.60；在此基础上，TGAT+AE的F1分别提高到87.29和92.70，GraphMixer+AE分别达到88.32和92.55，说明图结构越复杂，动态图对结构信号的学习收益越大。在此之上，DGCEC-MAD通过节点内存累积历史并以融合边特征的图注意力刻画容器拓扑变化，进一步提升性能：在MallService上Precision、Recall、F1相比GraphMixer+AE分别提升2.88、4.17、3.49；在TrainTicket上相比TGAT+AE的Precision与F1分别提升0.73和0.4（较GraphMixer+AE的F1高0.56），体现在复杂场景中的更高精确性与稳定性。

3.4 训练耗时实验

为评估各方法的训练代价，本文对各方法统计单轮（one-epoch）平均耗时，结果如表2所示：

从结果看，GraphMixer+AE由于主体为MLP、算子轻量，单轮训练最短；DCADAnomaly得益于其特征提取过程的简洁性，并且未引入序列建模等复杂结构，整体模型结构较为简单，因此训练速度较

表2 one-epoch平均耗时

方法	单轮训练平均耗时(秒)
OC-SVM	—
TopoMAD	51.54
TraceAnomaly	26.07
TraceGra	31.12
DCADAnomaly	18.21
DyRep+AE	27.78
TGAT+AE	30.35
GraphMixer+AE	**17.37**
DGCEC-MAD	27.13

快；TopoMAD 由于叠加多层 GCN、LSTM 与 VAE，计算图复杂且序列依赖强，因而最慢。DGCEC-MAD 的单轮耗时为 27.13s，与 DyRep+AE（27.78s）几乎相当；快于 TGAT+AE 和 TraceGra；略慢于 TraceAnomaly、DCADAnomaly 以及 GraphMixer+AE。DGCEC-MAD 的主要时间开销来自内存状态更新与图注意力，但在准确率与动态场景鲁棒性上的提升使该训练代价可接受。注：OC-SVM 为基于核的单类方法，不涉及深度网络训练的迭代优化，故未统计单轮训练耗时。

3.5 容器迁移实验结果与分析

为进一步检验不同方法面对容器迁移场景中的适配性，使用基于 MallService 的原数据集所训练好的模型应用于变更数据集进行测试并统计结果，实验结果如表3所示。

表3 容器迁移实验结果

方法	原数据集			变更数据集		
	Precision	Recall	F1	Precision	Recall	F1
OC-SVM	84.30	47.88	61.07	23.11	**100**	37.55
TopoMAD	85.76	68.58	76.21	23.97	76.98	36.56
TraceAnomaly	71.32	74.92	73.07	53.13	98.93	69.13
TraceGra	73.27	75.51	74.37	28.57	58.06	38.30
DCADAnomaly	82.14	62.16	70.77	68.57	43.64	53.33
DGCEC-MAD	**88.68**	**95.17**	**91.81**	**72.04**	70.72	**71.37**

OC-SVM 方法与 TraceAnomaly 在容器迁移后的 Precision 急剧下降至 23.11% 和 53.13%，尽管 Recall 达到 100% 和 98.93%，但由于高误报，整体 F1 值偏低，这是由于微服务环境发生容器迁移等动态变化时，尽管模型在训练时能够识别出与训练数据相似的正常模式，但迁移后的新正常模式会偏离已学习的边界，导致大量新的正常实例被错误地标记为异常。TopoMAD 方法同样在有变更数据集上的表现显著降低，Precision 降至 23.97%，F1 值仅为 36.56%。这归因于其依赖于静态拓扑信息，无法动态适应容器迁移导致的拓扑结构变化。TraceGra 方法在容器迁移后的检测性能也显著下滑，Precision 仅为 28.57%，Recall 下降至 58.06%，F1 值为 38.30%。尽管 TraceGra 采用了图神经网络与 LSTM 对时空特征进行建模，但在面对服务容器的动态迁移时，其模型未能有效捕捉到新的交互模式和性能波动，导

致检测性能大幅下降。DCADAnomaly 依赖于训练阶段的特征分布和有限的异常样本进行判别，受限于异常样本数量不足，模型难以全面学习各种异常和新的正常行为，迁移后部分正常实例容易被误判为异常，导致 Recall 和 F1 下降。相比之下，DGCEC-MAD 通过融合多种数据，构建了反映微服务动态交互的动态图模型，代码执行上下文特征反映服务的业务逻辑，各个服务模块的功能相对单一稳定，因此该特征不会因容器迁移而发生较大改变。模型在无需重新训练的情况下，依然能够应对容器迁移，保持高效的异常检测性能，在有变更数据集中依然维持高水准的表现。

3.6 消融实验分析

为验证DGCEC-MAD方法各组件的有效性，对 MallService 数据集设计了系统性的消融实验。具体设置包括：（1）移除双重负采样策略，仅使用随机节点负采样；（2）完全移除代码执行上下文特征；（3）将代码执行上下文的语义编码分别替换为 TF-IDF 和 Word2Vec；（4）将 KPI 与代码执行上下文在动态图学习阶段提前融合；（5）将标准自编码器替换为变分自编码器（Variational Autoencoder，VAE）[30]。实验结果如表 4 所示。

从实验结果可以看出，移除代码执行上下文特征对结果影响较为显著，F1 值从 91.81% 骤降至 72.37%，降幅高达 19.44 个百分点，这充分说明了代码执行上下文在微服务异常检测中的关键作用。在代码执行上下文编码方式对比中，TF-IDF 编码仅达到 75.14% 的 F1 值，Word2Vec 编码提升至 81.13%，而本文采用的 Sentence-BERT 编码实现了 91.81% 的最佳性能，体现了预训练模型对深度语义理解的优势。特征融合策略方面，早期融合导致 F1 值下降至 83.53%，相比分阶段融合策略下降了 8.28 个百分点，验证了 KPI 通常易于学习且变化模式相对简单，若与结构行为特征混合训练，会掩盖微服务调用模式中的细粒度异常特征，分阶段学习有助于分别建模结构行为与性能表现，避免特征混杂带来的干扰。双重负采样策略相比传统随机采样提升了 5.97 个百分点，表明构造具有挑战性的负样本有助于提升模型对图结构的深入学习。自编码器架构选择上，VAE 虽然性能接近，但其 KL 散度约束不适合微服务系统的异构特性。实验证实了 DGCEC-MAD 各组件设计的合理性，特别是代码执行上下文在异常检测中具有一定价值。

表 4 消融实验结果

方法	Precision	Recall	F1
w/o 双重负采样策略	83.24	88.61	85.84
w/o 代码执行上下文	72.02	72.72	72.37
w/ TF-IDF 编码	73.87	76.46	75.14
w/ Word2Vec 编码	73.89	89.93	81.13
w/ KPI 早期融合	74.30	**95.37**	83.53
w/ VAE	88.10	93.98	90.95
DGCEC-MAD	**88.68**	95.17	**91.81**

3.7 超参数实验

3.7.1 损失函数系数选取实验

为确定动态图学习阶段链接预测损失系数 α 与重构误差系数 β 的最优权重配比。设计了五组参数对[0.1:0.9，0.3:0.7，0.5:0.5，1:1，2:1] 对 MallService 数据集进行实验，结果如图 3(a)所示：

结果表明，两个学习目标需要适当平衡，既要充分学习微服务调用的动态图特征，又要保证异常检测任务的有效性。当 α=0.3，β=0.7 时模型达到最佳性能，F1 值

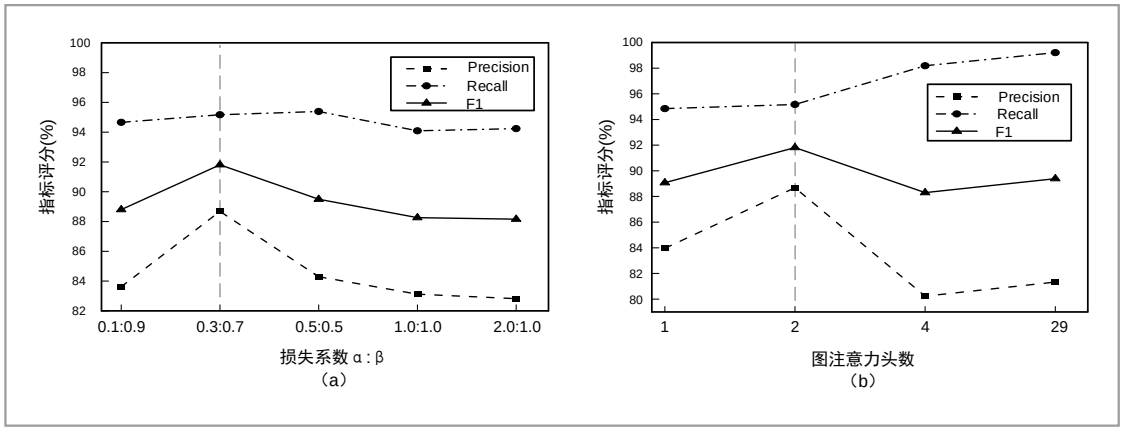

图3 超参数实验

为91.81%。过小的α值（如0.1）会削弱动态图学习效果，而过大的α值（如1.0以上）则会影响异常检测性能。

3.7.2 注意力头数实验

为确定图注意力网络的最优多头数量，进行了注意力头数的超参数实验，实验结果如图3(b)所示。结果表明，单头注意力由于表达能力有限，F1值仅为89.07%。随着多头数量增加到4和29，虽然Recall有所提升，但Precision显著下降，导致整体F1值降低。这是因为过多的注意力头会引入更多学习参数，模型可能会过多关注非主要特征，从而带来噪声，降低其判别能力。另一方面，过多的头数会进一步提高模型训练的开销。当多头数量为2时模型达到最佳性能，F1值为91.81%，既能捕获不同维度的特征依赖关系，又能保持模型的稳定性和准确性。

4 总结与不足

本文提出了一种基于动态图与代码执行上下文感知的微服务异常检测方法DGCEC-MAD。以带时间戳的有向边事件建模服务调用，其中特征由插桩采集的类名、方法名、参数与返回值等代码执行上下文经预训练语言模型编码得到。在动态图学习上，采用时间批处理与余弦时间编码，结合内存更新搭配结合时间与边特征的图注意力聚合，持续刻画容器迁移与调用关系的演化。在训练目标上，设计了融合边特征的链接预测并引入双重负采样，使节点表示更贴近真实调用语义与结构模式。实验结果表明，DGCEC-MAD在多组数据上相较基线在检测效果上有一定提升，在容器迁移、扩缩容等动态场景中仍保持稳定性能；消融研究验证了代码执行上下文对方法级调用可分性的提升，能够缓解仅依赖耗时统计导致的分布重叠问题。尽管如此，本文在异常根本原因分析方面尚未深入研究。未来的工作将致力于对异常事件进行更为细粒度的分析，明确异常产生的根本原因，为微服务系统的自动化运维和故障诊断提供更加有力的支持。

参考文献：

[1] 梅宏，杜小勇，金海，等. 大数据技术前瞻[J].

大数据, 2023, 9(1): 1-20.
MEI H, DU X Y, JIN H, et al. Big data technologies forward-looking[J]. Big Data Research, 2023, 9(1): 1-20.

[2] 王璐, 姜宇轩, 李青山, 等. 微服务故障检测研究综述[J]. 计算机学报, 2023, 46(11): 2342-2369.
Wang L, Jiang Y X, Li Q S, et al. A Survey on Microservice Fault Detection [J]. Chinese Journal of Computers, 2023, 46(11): 2342-2369.

[3] Pahl C. Containerization and the paas cloud[J]. IEEE Cloud Computing, 2015, 2 (3): 24-31.

[4] Chen J, Liu F, Jiang J, et al. TraceGra: A trace-based anomaly detection for microservice using graph deep learning [J]. Computer Communications, 2023, 204: 109-117.

[5] Liu P, Xu H, Ouyang Q, et al. Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks[C]// 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE). IEEE, 2020: 48-58.

[6] He Z, Chen P, Li X, et al. A spatiotemporal deep learning approach for unsupervised anomaly detection in cloud systems[J]. IEEE Transactions on Neural Networks and Learning Systems, 2020, 34(4): 1705-1719.

[7] Khodabandeh G, Ezaz A, Babaei M, et al. Utilizing graph neural networks for effective link prediction in microservice architectures[C]// Proceedings of the 16th ACM/SPEC International Conference on Performance Engineering. 2025: 19-30.

[8] Michelucci U. An introduction to autoencoders[J]. arXiv preprint arXiv: 2201.03898, 2022.

[9] Chen Y, Qian J, Saligrama V. A new one-class SVM for anomaly detection [C]// 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE, 2013: 3567-3571.

[10] Breunig M M, Kriegel H P, Ng R T, et al. LOF: identifying density-based local outliers[C]//Proceedings of the 2000 ACM SIGMOD international conference on Management of data. 2000: 93-104.

[11] Ikotun A M, Ezugwu A E, Abualigah L, et al. K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data [J]. Information Sciences, 2023, 622: 178-210.

[12] Liu F T, Ting K M, Zhou Z H. Isolation forest[C]// 2008 eighth IEEE International Conference on Data Mining. IEEE, 2008: 413-422.

[13] Huang T, Chen P, Li R. A semi-supervised VAE based active anomaly detection framework in multivariate time series for online systems[C]//Proceedings of the ACM Web Conference 2022. 2022: 1797-1806.

[14] Nguyen H X, Zhu S, Liu M. A survey on graph neural networks for microservice-based cloud applications[J]. Sensors, 2022, 22(23): 9492.

[15] Hochreiter S, Schmidhuber J. Long short-term memory[J]. Neural computation, 1997, 9(8): 1735-1780.

[16] Kipf T N, Welling M. Semi-supervised classification with graph convolutional networks[C]// International Conference on Learning Representations (ICLR). 2017

[17] Jing Y, Wang H, Chen J, et al. Dual-channel graph-level anomaly detection method based on multi-graph representation learning[J]. Applied Intelligence, 2025, 55(6): 500.

[18] Yang L, Chatelain C, Adam S. Dynamic

graph representation learning with neural networks: A survey[J]. IEEE Access, 2024, 12: 43460-43484.

[19] Trivedi R, Farajtabar M, Biswal P, et al. Dyrep: Learning representations over dynamic graphs[C]// International Conference on Learning Representations (ICLR). 2019

[20] Xu D, Ruan C, Korpeoglu E, Kumar S, Achan K. Inductive representation learning on temporal graphs[C]// International Conference on Learning Representations (ICLR). 2020

[21] Rossi E, Chamberlain B, Frasca F, et al. Temporal graph networks for deep learning on dynamic graphs[J]. arXiv preprint arXiv:2006.10637, 2020.

[22] Cong W, Zhang S, Kang J, et al. Do we really need complicated model architectures for temporal networks? [C]// The Eleventh International Conference on Learning Representations (ICLR). 2023

[23] Wang W, Wei F, Dong L, et al. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers[J]. Advances in neural information processing systems, 2020, 33: 5776-5788.

[24] Reimers N, Gurevych I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks[C]// Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). Association for Computational Linguistics, 2019: 3982.

[25] Chung J, Gulcehre C, Cho K H, et al. Empirical evaluation of gated recurrent neural networks on sequence modeling [J]. arXiv preprint arXiv: 1412.3555, 2014.

[26] 黄少年, 彭永涛, 文沛然, 等. 融合注意力协同和对比学习的跨模态突发事件识别方法[J]. 智能科学与技术学报, 2025.
HUANG S N, PENG Y T, WEN P R, et al. A Cross-modal emergency recognition method integrating attentional collaboration and contrastive learning[J]. 2025.

[27] Veličković P, Cucurull G, Casanova A, et al. Graph attention networks[C]// International Conference on Learning Representations (ICLR). 2018

[28] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition [C]// Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[29] Xu J, Sun X, Zhang Z, et al. Understanding and improving layer normalization[J]. Advances in neural information processing systems, 2019, 32.

[30] Kingma D P, Welling M. Auto-encoding variational bayes[J]. arXiv preprint arXiv:1312.6114, 2013.

作者简介

顾加伟（1998– ），男，昆明理工大学信息工程与自动化学院硕士研究生，主要研究方向为云计算、故障诊断、大数据分析等。

姜瑛（1974– ），女，博士，昆明理工大学信息工程与自动化学院教授，主要研究方向为软件质量保证与测试、云计算、大数据分析、智能软件工程。

杨天晴（1991– ），男，昆明理工大学信息工程与自动化学院博士研究生，主要研究方向为云计算、故障诊断、大数据分析等。

（CCF学生会员，会员号：Z1739G）

收稿日期: XXXX-XX-XX

通信作者: 姜瑛，jy_910@163.com

基金项目: 国家自然科学基金资助项目(No. 62162038, No. 61462049, No. 61063006, No. 60703116)；国家重点研发计划资助项目(No. 2018YFB1003904)；云南省应用基础研究计划重点基金项目(No. 2017FA033)

Foundation Items: The National Natural Science Foundation of China (No. 62162038, No. 61462049, No. 61063006, No. 60703116), The National Key Research and Development Program of China (No. 2018YFB1003904), Key Fund Project of Yunnan Province Applied Basic Research Program (No. 2017FA033)