IEEE Sensors Letters VOL. 7, NO. 10, OCTOBER 2023

IEEE Sensors Council

Sensor systems

TinyRCE: Multipurpose Forward Learning for Resource Restricted Devices

Danilo Pietro Pau[1*], Andrea Pisani[1**], Fabrizio Maria Aymone[1**], and Gianluigi Ferrari[2***]

[1]Department of Systems Research, Application, STMicroelectronics, 20041 Agrate Brianza, Italy
[2]IoT Lab, Department of Engineering and Architecture, University of Parma, 43124 Parma, Italy
[*]Fellow, IEEE
[**]Member, IEEE
[***]Senior Member, IEEE

Manuscript received 30 June 2023; revised 27 July 2023; accepted 8 August 2023. Date of publication 22 August 2023; date of current version 26 September 2023.

Abstract—The challenge of deploying neural network (NN) learning workloads on ultralow power tiny devices has recently attracted several machine learning researchers of the Tiny machine learning community. A typical on-device learning session processes real-time streams of data acquired by heterogeneous sensors. In such a context, this letter proposes Tiny Restricted Coulomb energy (TinyRCE), a forward-only learning approach based on a hyperspherical classifier, which can be deployed on microcontrollers and potentially integrated into the sensor package. TinyRCE is fed with compact features extracted by a convolutional neural network (CNN), which can be trained with backpropagation or it can be an extreme learning machine with randomly initialized weights. A forget mechanism has been introduced to discard useless neurons from the hidden layer, since they can become redundant over time. TinyRCE has been evaluated with a new interleaved learning and testing data protocol to mimic a typical forward on-tiny-device workload. It has been tested with the standard MLCommons Tiny datasets used for keyword spotting and image classification, and against the respective neural benchmarks. In total, 95.25% average accuracy was achieved over the former classes (versus 91.49%) and 87.17% over the latter classes (versus 100%, caused by overfitting). In terms of complexity, TinyRCE requires 22× less Multiply and ACCumulate (MACC) than SoftMax (with 36 epochs) on the former, whereas it requires 5× more MACC than SoftMax (with 500 epochs) for the latter. Classifier complexity and memory footprint are marginal w.r.t. the feature extractor, for training and inference workloads.

Index Terms—Sensor systems, on-device learning, Extreme learning machines (ELMs), feature extraction, hyperspherical classifier, keyword spotting (KWS), on-tiny-device learning, Tiny machine learning (TinyML).

I. INTRODUCTION

Supervised artificial intelligence (AI) and Tiny machine learning (TinyML) nowadays are widely deployed approaches. Bianco et al. [1] benchmarked multiple deep neural networks (DNNs or NNs) reporting accuracy, memory usage, complexity, and latency. As the complexity of the DNNs increased, by addressing more challenging problems, the DNNs required bigger model size, more memory, and computational capabilities. The deployment of complex DNNs on tiny devices and their use to process sensor-generated data streams can also be affected by a decrease in inference accuracy w.r.t. the training phase [2]. This is due to numerous causes. The consequence is that continuous updates and retraining of the DNN models are required over time. The de-facto standard training method, integrated by many deep learning frameworks [1], is based on backpropagation (BP) and stochastic gradient descent. Retraining with such a complex algorithm requires powerful computational and storage assets. Unfortunately, when using tiny devices, continuous learning (CL) is very costly on them [3]. Moreover, BP is a type of sequential learning, which can be heavily affected by catastrophic forgetting (CF) [4] compromising the CL objectives. Thus, most tiny devices pushed sensors data to the cloud, which is capable to run highly asset-demanding AI workloads. Although cloud-based solutions [5] feature more resources than the tiny devices, they still account for many disadvantages. Privacy, security of user data, latency, communication bandwidth, and power consumption are big concerns for the TinyML community.

Corresponding author: Danilo Pietro Pau (email: danilo.pau@st.com).
Associate Editor: F. Falcone.
Digital Object Identifier 10.1109/LSENS.2023.3307119

Therefore, there is the opportunity to rethink supervised learning procedures and extend the state of the art beyond BP by conceiving new on-tiny-device learning algorithms. An approach to on-tiny-device CL without using BP is proposed by this letter, which is organized as follows. Section II introduces the problem statement and the requirements to be fulfilled by any solution. Section III reviews the existing related works and reports their limitations. Section IV summarizes the datasets used to shape the case studies for experimenting the learning and testing workloads. Section V explains the proposed solution, highlights differences with respect to the previous works, and describes how the on-device learning (ODL) field was approached. The experimental results are reported in Section VI. Section VII analyzes the complexity of the proposed solution with respect to its deployment on microcontrollers (MCUs). Finally, Section VIII concludes this letter.

II. PROBLEM STATEMENT AND REQUIREMENTS

The research question set by this letter is: can incremental learning of multiple categories happen, totally online without CF, without K-fold BP, and still be deployable on tiny devices? Therefore, the requirements, to solve such a problem, were set as given in Table 1. They shall be fulfilled by the proposed or existing (if any) solution.

III. RELATED WORKS

A. ODL and Extreme Learning Machines (ELMs)

CL DNNs aim to learn different tasks without retraining from scratch nor forgetting previous knowledge. CF [4] is the most common issue

TABLE 1. Requirements for Any Tiny ODL Solution

No.	Requirements
1.	Real-time forward learning.
2.	No BP.
3.	Deployable on tiny MCU, optionally in the sensor fitting into the embedded memory.
4.	Capable of performing multi-class classification.
5.	Capable of interleaving learning with inference workloads.
6.	Requiring the minimum temporary storage of sensor data.
7.	Featuring limited latency to run inference workloads.

affecting the performances of CL models. A related issue affecting Tiny ODL models is concept drift [6]. Pau and Ambrose [7] reported a state-of-the-art review for both ODL and ML on tiny devices and summarized the importance of the challenges faced.

ELMs are DNNs with randomly initialized weights. Only the ELMs' final dense layer is optimized through linear methods. Huang et al. [8] introduced the concept of ELM on edge devices. ELM was introduced by Huang et al. [9] for regression and multiclass classification applications. Zhang et al. [10] proposed a survey for on-line ELM.

B. Introduction to Restricted Coulomb Energy (RCE) and Hyperspherical Classifiers

RCE classifiers [11] are a branch of hyperspherical classifiers [12]. They are composed of three layers: input, hidden, and output. Hidden neurons are dynamically initialized hyperspheres defined by center points that are samples of the feature space and a default radius. Each hidden neuron is fully connected to the neurons of the input layer, while it is connected to only one output neuron. New hidden neurons are instantiated during the learning phase if some input feature vectors do not fall inside existing hyperspheres, according to a pairwise metric, usually the Euclidean distance. Multiple hidden neurons could ambiguously fire at once. The radius of hidden neurons that misfire and lead to incorrect predictions is reduced so that the corresponding hyperspheres cease to contain the misclassified inputs. Should a hidden neuron feature a very small radius, it can be considered redundant and therefore be removed. Unfortunately, in a naive implementation, that removal would not happen. Sui et al. [13] presented a DNN to segment hands appearing into images. The skin-colored pixels were classified and segmented using the RCE. A new approach to reduce the neuron's radius has been introduced to lower the processing. However, it did not face how to remove the redundant neurons. Fanizzi et al. [14] proposed an RCE inductive classifier, which was more efficient than other similar approaches. All the above approaches were not meant for MCU deployment. They adopted the traditional offline training methods and did not perform ODL on MCU.

C. MLCommons Benchmark for Keyword Spotting (KWS) and Image Classification (IC)

The MLPerf Tiny benchmark [15] is a collection of TinyML benchmark models. They were created thanks to the collaborative effort of more than 50 affiliates, spanning from leading companies to academic research groups. It focuses on four use cases: KWS; visual wake words; IC; and anomaly detection. In particular, the KWS NN model is the depthwise separable convolutional neural network (DS-CNN), 92.09% accurate over the Speech Commands v2 dataset [16], within 36 epochs, by means of BP. The IC benchmark is an implementation of ResNetv1 [17] NN, 86.5% accurate over CIFAR-10 [18] dataset within 500 epochs by means of BP.

IV. DATASETS

KWS and IC datasets shaped the case studies considered by this work. Speech commands v2 [16] is a widely used speech recognition dataset that represents the former use case. It is available under the Creative Commons BY license, and is part of the TensorFlow Datasets[1] library. It provides 105 829 audio recordings of single word utterances, recorded by volunteers with their own devices and shared online in a WAV format with a sampling rate of 16 kHz. They are divided into training (85 511 samples), validation (10 102), and testing (4890) subsets. The data are divided in 12 classes: ten are associated with relevant keywords. The two remaining ones are associated with silent recordings and nonrelevant keywords labeled as "Unknown." CIFAR-10 [18] is a dataset of low-res (32px $\times$ 32px) RGB images collected from the Internet. It consists of 60 000 images of vehicles and animals divided in ten classes (6000 images per class), encoded as NumPy matrices. The dataset is divided into five training batches and one testing batch, of 10 000 images each. One task of this work was to compare the proposed solution against the MLCommons/Tiny benchmark NNs [15]. Therefore, the same preprocessing pipeline was used. In order to mimic the incremental introduction of new classes over time, the data tensors were rearranged by label and sequentially presented to the classifier, using 20% of the data of each class as training samples and 80% as testing samples.

V. PROPOSED ODL ALGORITHM

TinyRCE NN is a hyperspherical classifier variant proposed by this work. The raw data are input to the CNN feature extractor (FE). Convolution operations into CNNs extract abstracted features from raw data tensors. Thus, a CNN topology stripped from its final SoftMax layer is considered to be the FE. It was trained offline, using BP. A random ELM-style initialization for the FE was performed, and it impacted negatively the classifier's accuracy since it was not able to spatially separate features of the different classes. The extracted features then fed the TinyRCE classifier. For the KWS use case, the MLCommons benchmark DS-CNN was adopted as FE after removing its classification layer. For IC, the MLCommons benchmark ResNetv1 NN was used.

TinyRCE was designed to address all the requirements in Table 1. Only the hidden neurons of TinyRCE were changed by the ODL procedure. Data classes were presented sequentially to mimic the CL setup. TinyRCE modified its hidden and output layers by processing the features extracted by the FE. CL and the dynamic instantiation of new neurons into TinyRCE could quickly saturate the fixed size of the MCU embedded memory, if handled in an uncontrolled way. Therefore, a culling mechanism was introduced. It worked by assigning to each hidden neuron an unsigned 8-b integer age value and a floating point 32-b reliability value. At time t_0, said values were set to 0 for each instantiated hidden neuron. The age value was incremented by 1 every time a neuron fired. The reliability score s_j was updated accordingly to (1). Note that $\hat{y}_t$ represents a label prediction done by the model for sample t, why y_t represents the true label of the same sample:

$$s_j^{\text{new}} = \begin{cases} s_j^{\text{old}} + \frac{R_j - \text{dist}\,(h_j, x_t)}{R_j} & \text{if } \hat{y}_t = y_t \\ s_j^{\text{old}} - \frac{\text{dist}\,(h_j, x_t)}{R_j} & \text{if } \hat{y}_t \neq y_t. \end{cases} \quad (1)$$

In (1), the increment and decrement were different: the increment's numerator favored the correct predictions that involved inputs located close to the hypersphere's center instead of those involving inputs

[1][Online]Available: https://www.tensorflow.org/datasets?hl=en

located near its border. The decrement formula did the opposite. When the feature fell inside the influence region of a hidden neuron j that matched the ground truth annotation, s_j was increased; otherwise, decremented. This value enabled pruning of redundant neurons, which was triggered if a threshold was exceeded. The threshold depended on the maximum number of hidden neurons MCU could fit into its RAM memory. When pruning the redundant neurons, both the age score and the reliability score were used. Neurons that were deemed redundant were those with the least number of firings and the lowest reliability.

It is well known that hyperspherical classifiers produce feature spaces that overlap with each other's classes. This implies to activate simultaneously several hidden neurons associated with different classes. Furthermore, at the end of the CL workflow, the hyperspheres associated with the last class dominated the feature space with respect to the other classes' hyperspheres, since they were exposed to the least corrections over time. To avoid these issues, the dynamic instantiating of the hyperspheres was modified w.r.t. the classic RCE algorithm. Also, a method to make the model more robust to noise in the data was introduced.

The dynamic instantiating of new hyperspheres was modified as follows. If an input vector was positioned in a location of the feature space not yet covered by any hypersphere, the radius of the new hypersphere to instantiate (denoted by R^*) was determined dynamically instead of being set by a user-defined standard value $\overline{R}$: since the overlapping of hyperspheres with different labels led to incorrect predictions, the model checked if the newly instantiated hypersphere overlapped with other ones that were differently labeled. If this was the case, then the radius of the new hypersphere was set to the biggest possible value that caused no such overlap. Otherwise, the standard value was kept. This behavior is summarized in (2), which defines R^*:

$$R^* = \begin{cases} \overline{R} & \forall j, \operatorname{dist}(h_j, x_t) > \overline{R} \\ \min\limits_j(\operatorname{dist}(h_j, x_t)) - R_j & \text{otherwise.} \end{cases} \tag{2}$$

Finally, to achieve better noise resilience, the implemented scoring system also tied in to the neuron resizing policy. A hypersphere was not supposed to be resized if its misfiring was caused by a data outlier. An outlier was defined as a wrong prediction fired by a hypersphere whose reliability score was in the first quartile among all existing hyperspheres. Assuming the input was recognized as data outlier, the misfiring of a hypersphere did not cause its resizing, only the decrease of its reliability score.

In the case of IC, Bayesian optimization [19] was applied to find set of hyperparameters, which maximized the testing accuracy. The variables to optimize were: the standard radius $\overline{R}$; the pruning threshold; the train-test split to divide the dataset; a growth law that determined how $\overline{R}$ would change from one class to the next during CL, chosen among a constant function, a linear and an exponential growth; a weight w, which was part of the target metric to optimize. This target metric is described in (3), where acc_x stands for accuracy, and N represents the number of classes:

$$\text{score} = \text{acc}_{\text{test set}} - w \min\left\{0, \left(0.9 - \frac{1}{N}\sum_{k=1}^{N} \text{acc}_{\text{class } k}\right)\right\}. \tag{3}$$

VI. EXPERIMENTS

For both KWS and IC, the FE produced 64-dimension features. Randomly initialized ELM versions performed poorly. On the contrary, the off-the-shelf trained topologies from MLCommons Python scripts achieved well separated feature spaces.

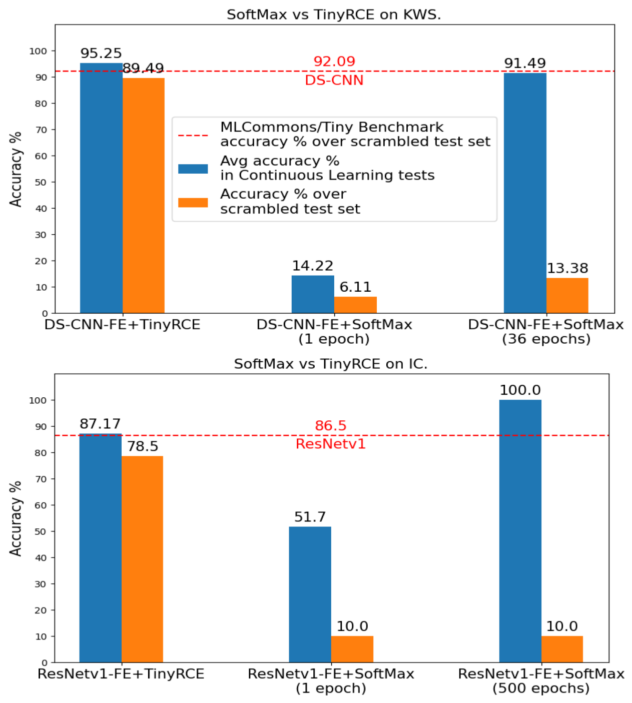

Fig. 1. Experimental results on KWS and IC cases.

TinyRCE proved to outperform the SoftMax over the ODL procedure. For KWS, TinyRCE was 95.25% accurate (average on tests) w.r.t. 91.49% of the SoftMax classifier trained with 36 epochs. SoftMax performed very poorly (average accuracy 14.22%) if trained using only 1 epoch w.r.t. TinyRCE. Over the standard test set, TinyRCE performed comparably w.r.t. the benchmark network trained with BP. For IC, SoftMax performed even worse: if trained with 500 epochs, it totally overfit the training data, thus achieving 100% training accuracy while performing as a random predictor if deployed on the test set. On the contrary, TinyRCE's average accuracy over the single classes (87.17%) was closer to its accuracy on the scrambled test set (78.5%). Histograms displaying the results for both use cases are shown in Fig. 1.

VII. COMPLEXITY ANALYSIS FOR TINY DEVICES

Complexity was measured in Multiply and ACCumulate (MACC) operations. The equations to estimate the complexity for TinyRCE's inference and learning phases are described in (4) and (5), respectively

$$\text{MACC}_{\text{inference}} = h \times [(n \times 5) + 10] \tag{4}$$

h is the number of hidden neurons, and n is the dimensionality of the feature vectors;

$$\text{MACC}_{\text{learning}} = \{h \times [(n \times 5) + 10]\} \times (N \times E). \tag{5}$$

N is the number of training data, and E is the number of epochs in the worst case.

The complexity analysis was performed with two off-the-shelf MCUs: NUCLEO-H743ZI2 and B-U585I-IOT02A. The former runs at 480 MHz, with 1024 KiB of embedded RAM and 2048 KiB of embedded Flash, whereas the latter runs at 160 MHz, with 786 KiB RAM and 2048 KiB Flash both on chip. For the FEs, the analysis consisted in evaluating the MACCs needed by each layer of the NN for training and inference operations (e.g., forward pass and backward pass). During inference, RAM stored the activation buffers, whereas Flash stored the

TABLE 2. Complexity Profiling and Memory Footprint Analysis for the KWS Use Case

Metrics		DS-CNN-FE_BP	SoftMax (1 epoch)	SoftMax (36 epochs)	TinyRCE
MACC (M)	Training	2.429E+5	**3.494**	125.8	5.643
	Inference	2.664	**9.480E-4**	**9.480E-4**	3.960E-3
RAM (KiB)	Training	866.3	**64.0**	**64.0**	**64.0**
	Inference	**64.0**	**64.0**	**64.0**	**64.0**
FLASH (KiB)	Inference	**3.120**	**3.120**	**3.120**	3.216
Inference time (ms)	NUCLEO-H743ZI2	-	**38.05**	**38.05**	38.10
	B-U585I-IOT02A	-	**162.96**	**162.96**	163.14

The best values for each metric appear in bold.

TABLE 3. Complexity Profiling and Memory Footprint Analysis for the IC Use Case

Metrics		ResNet-FE	SoftMax (1 epoch)	SoftMax (500 epochs)	TinyRCE
MACC (M)	Training	92.75E+6	**2.100**	1,050.0	5,266.8
	Inference	12.5	**7.90E-4**	**7.90E-4**	9.24E-2
RAM (KiB)	Training	455.12	**196.60**	**196.60**	**196.60**
	Inference	**196.60**	**196.60**	**196.60**	**196.60**
FLASH (KiB)	Inference	**2.600**	**2.600**	**2.600**	2.680
Inference time (ms)	NUCLEO-H743ZI2	-	**123.61**	**123.61**	124.51
	B-U585I-IOT02A	-	**527.03**	**527.03**	530.86

The best values for each metric appear in bold.

trainable parameters. During the training phase, the parameters shall be continuously updated, therefore they were stored in RAM. Hence, peak RAM utilization for all the topologies was limited by the memory bottleneck imposed by the forward pass through the frozen FEs. Total inference times were evaluated using the STM32Cube.AI Developer Cloud[2] service. For both KWS and IC, the training of the FE with BP dominated complexity and memory footprint. The difference between the two classifiers was marginal w.r.t. the FE they had in common. For KWS, peak RAM was 866.3 KiB (estimated) for training and 64 KiB for inference. The Flash memory required was 3.12 KiB for the DS-CNN and 3.216 KiB for DS-CNN-FE_BP+TinyRCE. For IC, peak RAM was 455.12 KiB for training and 196.60 KiB for inference. The Flash memory footprint was 2.68 KiB (estimated).

VIII. CONCLUSION

This letter introduced TinyRCE, which performed forward-only incremental classification learning on MCU. No BP was required, except for the FE training. The FEs were off-the-shelf solutions from MLCommons Tiny working group. A specific protocol for both training and testing procedures was introduced, mimicking the streaming acquisition of raw sensor data. TinyRCE proved to be more accurate than the MLCommons benchmark models trained for multiple epochs on the ODL training–testing workload. Complexity and memory footprint were dominated by the offline-trained FEs. Future developments will be focused on devising an ELM FE, in order to eliminate the BP procedure applied to the FE. Further RAM and Flash reduction using low bit-depth features could ease the deployability into the sensor itself. Also, new use cases regarding different data types (e.g., time of flight) could be investigated.

[2][Online]Available: https://stm32ai-cs.st.com

REFERENCES

[1] S. Bianco, R. Cadène, L. Celona, and P. Napoletano, “Benchmark analysis of representative deep neural network architectures,” *IEEE Access*, vol. 6, pp. 64270–64277, 2018.

[2] D. Vela, A. Sharp, R. Zhang, T. Nguyen, A. Hoang, and O. S. Pianykh, “Temporal quality degradation in AI models,” *Sci. Rep.*, vol. 12, Jul. 2022, Art. no. 11654.

[3] M. Delange et al., “A continual learning survey: Defying forgetting in classification tasks,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 7, pp. 3366–3385, Jul. 2019.

[4] P. Kaushik, A. Gain, A. Kortylewski, and A. Yuille, “Understanding catastrophic forgetting and remembering in continual learning with optimal relevance mapping,” in *Proc. 5th Workshop “Meta-Learn.” 35th Conf. Neural Inf. Process. Syst.*, 2021, pp. 1–22.

[5] F. Samie, L. Bauer, and J. Henkel, “From cloud down to things: An overview of machine learning in Internet of Things,” *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4921–4934, Jun. 2019.

[6] Y. Kadwe and V. Suryawanshi, “A Review on Concept Drift,” *IOSR J. Comput. Eng.*, vol. 17, no. 1, pp. 20–26, 2015.

[7] D. Pau and P. K. Ambrose, “A quantitative review of automated neural search and on-device learning for tiny devices,” Chips, vol. 2, no. 2, pp. 130–141, Jan. 2022.

[8] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, “Extreme learning machine: A new learning scheme of feedforward neural networks,” in *Proc. IEEE Int. Joint Conf. Neural Netw.*, 2004, vol. 2, pp. 985–990.

[9] G.-B. Huang, H. Zhou, X. Ding, R. Zhang, R. Zhang, and R. Zhang, “Extreme learning machine for regression and multiclass classification,” *IEEE Trans. Syst., Man, Cybern., Part B*, vol. 42, pp. 513–529, Apr. 2012.

[10] S. Zhang, W. Tan, and Y. Li, “A survey of online sequential extreme learning machine,” in *Proc. IEEE 5th Int. Conf. Control, Decis. Inf. Technol.*, 2018, pp. 45–50.

[11] M. Hudak, “RCE networks: An experimental investigation,” in *Proc. IJCNN-91-Seattle Int. Joint Conf. Neural Netw.*, 1991, pp. 849–854.

[12] M. H. Hassoun et al. *Fundamentals of Artificial Neural Networks*. Cambridge, MA, USA: MIT Press, 1995.

[13] C. Sui, N. M. Kwok, and T. Ren, “A restricted Coulomb energy (RCE) neural network system for hand image segmentation,” in *Proc. IEEE Can. Conf. Comput. Robot Vis.*, 2011, pp. 270–277.

[14] N. Fanizzi, C. d’Amato, and F. Esposito, “ReduCE: A reduced Coulomb energy network method for approximate classification,” in *Extended Semantic Web Conf.*, 2009, pp. 323–337.

[15] C. R. Banbury et al., “MLPerf Tiny benchmark,” in *Proc. 35th Conf. Neural Inf. Process. Syst. NeurIPS Datasets Benchmarks*, 2021.

[16] P. Warden, “Speech commands: A dataset for limited-vocabulary speech recognition,” 2018, *arXiv:1804.03209*.

[17] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. Comput. Vis. Pattern Recognit.*, 2015, pp. 770–778.

[18] “CIFAR-10 and CIFAR-100 datasets,”. Accessed: Aug. 23, 2023. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[19] R. Garnett, *Bayesian Optimization*. Cambridge, U.K.:Cambridge Univ. Press, 2023.