High-Precision Close-to-Analog Programming of PCM Cells as Devices for AiMC Edge-AI

A. Antolini[1*], A. Lico[1*], F. Zavalloni[1], L. Greco[1], R. Zurla[2], J. Bertolini[3], R. Vignali[4], L. Iannelli[4], E. Calvetti[3], M. Pasotti[3], A. Cabrini[4], E. Franchi Scarselli[1]
[1]ARCES-DEI, University of Bologna, Bologna, Italy; [2]STMicroelectronics, Pavia, Italy; [3]STMicroelectronics, Agrate Brianza, Italy; [4]University of Pavia, Pavia, Italy.

***Abstract*—One of the most appealing characteristics of Phase-Change Memory (PCM) technology is the ability to store analog quantities as conductances. To fruitfully exploit this feature in Analog in-Memory Computing (AiMC) applications, high-precision programming algorithms are required. In this paper, a method to reach a fairly arbitrary number of analog levels in PCM cells is proposed. The algorithm has been validated on an AiMC prototype, designed and fabricated using a 28-nm FD-SOI process by STMicroelectronics to compute 512×512 signed Matrix-Vector Multiplications (MVMs), achieving an Equivalent Number of Bits (ENOB) of 10.55, which improves state-of-the-art weights programming accuracy for PCM-based hardware accelerators.**

***Index Terms*—Analog in-Memory Computing (AiMC), Analog Programming, Matrix-Vector Multiplication (MVM), Phase-Change Memory (PCM), Deep Neural Network (DNN).**

I. Introduction

ANALOG in-Memory Computing (AiMC) integrates processing directly in resistive non-volatile memories to perform mathematical operations, such as Matrix-Vector Multiplications (MVMs), which are particularly useful for Deep Neural Networks (DNNs) [1], [2]. This integration significantly reduces data transfer overhead, a major bottleneck in traditional architectures. Phase-Change Memory (PCM) can enhance AiMC by enabling analog programming of the stored MVM coefficients. PCM cells are typically written by means of iterative program-and-verify algorithms. However, the number of states that can be effectively stored inside the memory is limited by the finite tolerance of the programming algorithm and by the precision of the cell measurement chain employed for the verify step.

This work presents an analog programming method for PCM cells that grants fairly arbitrary number of bits per weight; the procedure is experimentally assessed on a 4M cells Ge-rich GST PCM-based AiMC core designed and fabricated in a 28-nm FD-SOI STMicroelectronics process. The paper is organized as follows: in Section II the analog programming procedure is detailed; Section III presents the experimental results, which are then compared with the state of the art.

II. Analog programming of PCM cells

A. PCM-based AiMC architectures

MVM $\mathbf{z} = \mathbf{W} \cdot \mathbf{x}$ can be executed within an ePCM array of dimensions $n \times m$ as illustrated in Fig. 1. Each matrix element $w_{i,j}$ is represented by conductance $g_{i,j}$ stored in one or more PCM devices. The input elements x_i of the vector $\mathbf{x} = [x_1, \ldots, x_n]$ are generally encoded in PWM intervals T_i, which are applied to the i-th wordline (WL) [3]. By biasing the j-th bitline (BL) with a voltage V_B and integrating the resulting current $i_{\mathrm{BL},j}$, the charge Q_j accumulated over the maximum integration time T^{MAX} supported by the ADC is

$$Q_j = \int_0^{T^{\mathrm{MAX}}} i_{\mathrm{BL},j}(t)dt = \sum_{i=1}^{n} Q_{i,j} = V_\mathrm{B} \sum_{i=1}^{n} g_{i,j} T_i \,, \quad (1)$$

which is then converted into a N-bit digital string by the corresponding ADC to generate the MVM output [4]

$$z_j := \left\lfloor \frac{Q_j}{Q_{\mathrm{FSR}}} 2^N \right\rfloor \,, \quad (2)$$

where Q_{FSR} corresponds to the ADC full scale range.

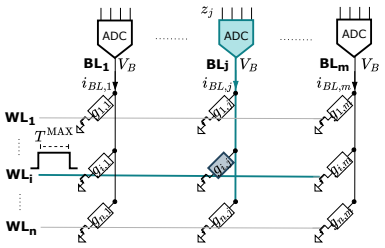

Fig. 1: Typical structure of PCM-based AiMC systems. The programming of cell $g_{i,j}$ requires proper row-column selection and is performed through the corresponding ADC, which provides the string $z_{i,j}$ after the application of a predefined activation interval T^{MAX}.

B. ADC-based cells programming

Weights programming in PCM cells involves the application of iterative sequences of current pulses to gradually modify cells conductance. Using for the verify step the same ADCs employed for the MVM computation ensures cells are programmed accounting for computation chain non-idealities (e.g., variability and non-linearity) [5]. The readout of a single PCM device conductance $g_{i,j}$ during programming can be assessed by activating uniquely the corresponding i-th WL with the maximum activation interval T^{MAX} (Fig. 1). Consequently, the j-th ADC outputs a digital string $z_{i,j}$ encoding $Q_{i,j}$, where

$$Q_{i,j} = V_B g_{i,j} T^{\mathrm{MAX}} \,. \quad (3)$$

* Equally contributing authors. Correspondance: alessio.antolini@unibo.it.

Since Q_{FSR} is chosen in relation to (1), Q_{FSR} is reasonably much higher than the maximum value $Q_{i,j}^{\mathrm{MAX}}$ of (3). Assuming the ADC to have N bits, the effective number of bits N_{eff} of $z_{i,j}^{\mathrm{MAX}}$, encoding $Q_{i,j}^{\mathrm{MAX}} = V_B g^{\mathrm{MAX}} T^{\mathrm{MAX}}$, is

$$N_{\mathrm{eff}} = N - \log_2 \frac{1}{\gamma}\,, \tag{4}$$

where $\gamma := Q_{i,j}^{\mathrm{MAX}}/Q_{\mathrm{FSR}}$. The value of $\gamma \leq 1$ quantifies the resolution loss of the programming output with respect to the available ADC N bits. In the specific case where Q_{FSR} corresponds to the maximum value of the MVM output (1), i.e., $Q_{\mathrm{FSR}} = n\, g^{\mathrm{MAX}} T^{\mathrm{MAX}}$, (4) becomes $N_{\mathrm{eff}} = N - \log_2 n$. Consequently, the reduction of programming precision increases also with the number n of cells per BL.

Since g^{MAX} is typically technology constrained and V_B cannot differ from the value employed for computations, an increase in γ can be achieved by extending the integration time T^{MAX} during the programming phase, i.e., by reducing the clock frequency that generates the PWM input signal. However, this solution prevents N_{eff} from going beyond N.

C. Accumulation for fairly-analog programming

To overcome previous limitation, we propose to iterate the computation of $Q_{i,j}$ and its conversion to $z_{i,j}$ at each programming step. The samples $z_{i,j}$ at each iteration are then accumulated externally into

$$z_{i,j}^{\mathrm{tot}} = \sum_{h=1}^{M} z_{i,j}[h]\,, \tag{5}$$

where M is the number of iterations. Hence, the overall effective amount of bits N'_{eff} of $z_{i,j}^{\mathrm{tot}}$ is

$$N'_{\mathrm{eff}} = N_{\mathrm{eff}} + \log_2 M = N - \log_2 \frac{1}{\gamma} + \log_2 M\,. \tag{6}$$

Consequently, if $M > 1/\gamma$, the possible $2^{N'_{\mathrm{eff}}}$ target values for $z_{i,j}^{\mathrm{tot}}$ can be increased even above the ADC N-bit resolution, so that a fairly analog programming of the PCM cells conductance is granted. It should be noted that the additional $\log_2 M$ bits in (6) convey information only if the peak-to-peak amplitude $Q_{i,j}^{pp}$ of the charge $Q_{i,j}$ (expressed in (3)) exceeds the ADC quantization step $Q_{\mathrm{LSB}} := Q_{\mathrm{FSR}}/2^N$, otherwise, the output would be converted to the same level in all steps h (Fig. 2(a)). As a result, the noise of $Q_{i,j}$ distributes their samples $z_{i,j}[h]$ across multiple quantization levels (Fig. 2(b)). This can be empirically verified with the sample amplitude $z_{i,j}^{pp}$.

D. Equivalent Number of Bits (ENOB)

Fig. 3 reports a sketch of a generic iterative program-and-verify algorithm of a PCM cell. The process starts with the definition of a target $\hat{z}_{i,j}$, which is then compared with $z_{i,j}^{\mathrm{tot}}$ to verify the state of the cell. If the programming error $\varepsilon := |\hat{z}_{i,j} - z_{i,j}^{\mathrm{tot}}|$ is less than the programming tolerance Δ_p, the process ends, otherwise a new programming attempt (typically consisting in the application of a proper current pulse) is performed. Thanks of the accumulation in the verify phase, $\hat{z}_{i,j}$ can range from 0 to $\hat{z}_{i,j}^{\mathrm{MAX}} := M\, z_{i,j}^{\mathrm{MAX}} = 2^{N'_{\mathrm{eff}}} - 1$.

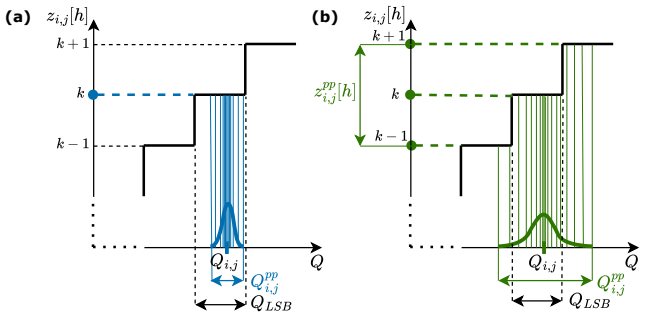

Fig. 2: Schematic representation of the variability on samples $z_{i,j}[h]$ noise to allow for bit expansion with accumulation; k is representative of the generic ADC output level. In (a) noise is not sufficient to grant ADC accuracy increase, whereas in (b) the noise level enhances the output variability.

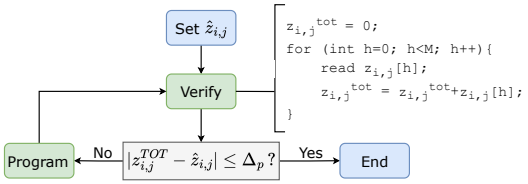

Fig. 3: Sketch of a generic programming algorithm for PCM cells with accumulation phase at each verify step.

The precision of the programming outcome of the MVM weights is quantified by the Equivalent Number of Bits (ENOB) [6], which is defined as

$$\mathrm{ENOB} := \log_2 \left(\frac{\hat{z}_{i,j}^{\mathrm{MAX}}}{\sigma_\varepsilon} \right) = N'_{\mathrm{eff}} - \log_2 \sigma_\varepsilon\,, \tag{7}$$

where σ_ε is the standard deviation of the programming error ε. The ENOB is increased by N'_{eff} (i.e., by the accumulation factor M) and lessened by the the weights error. Specifically, σ_ε can be estimated as $\sigma_\varepsilon \simeq \Delta_p/3$, where Δ_p is the programming tolerance set for the algorithm convergence. As a result, to account for the precision drop of the programming threshold Δ_p, N'_{eff} may be oversized with respect to the total amount of bits required by the specific computation task.

III. Experimental Results

The test vehicle utilized is a Ge-rich GST PCM-based AiMC macro fabricated in a STMicroelectronics 28-nm FD-SOI process, featuring a 4M-cell array (Fig. 4(a)). The AiMC macro executes a 512×512 signed matrix-vector multiplication (MVM) by leveraging 512 integrated ADCs to digitize the bitline (BL) charge, which is biased at $V_B = 0.1$ V, into an 11-bit sign-magnitude format. The input data x_i are encoded in an 8-bit sign-magnitude format. The signed MVM weights $w_{i,j}$ are stored in two cells, $g_{i,j}^{+}$ and $g_{i,j}^{-}$, whose conductance difference gives the absolute value of weights. Cells are programmed in PCM cells using dedicated write circuitry, and the programming algorithms are executed on an integrated microcontroller.

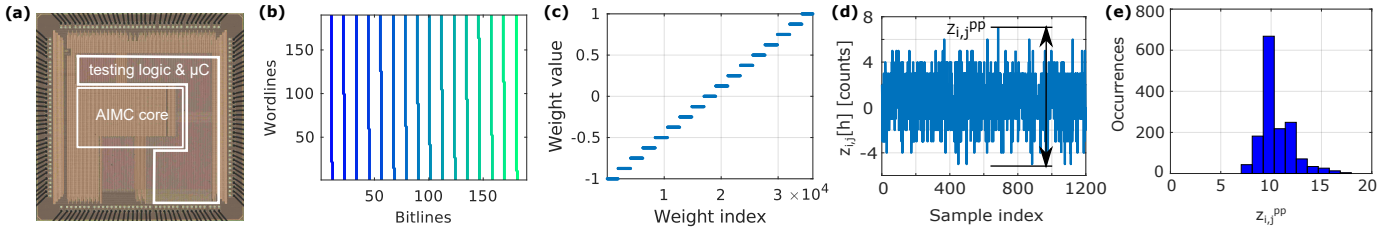

Fig. 4: (a) Micrograph of the employed AiMC macro. (b) Contour map and (c) coefficients distribution of the 190×190 programmed coefficients without accumulation ($M = 1$). (d) Example of measured charge samples $z_{i,j}$, and corresponding definition of $z_{i,j}^{pp}$. (e) Distribution of $z_{i,j}^{pp}$ over a 1.6k PCM cells.

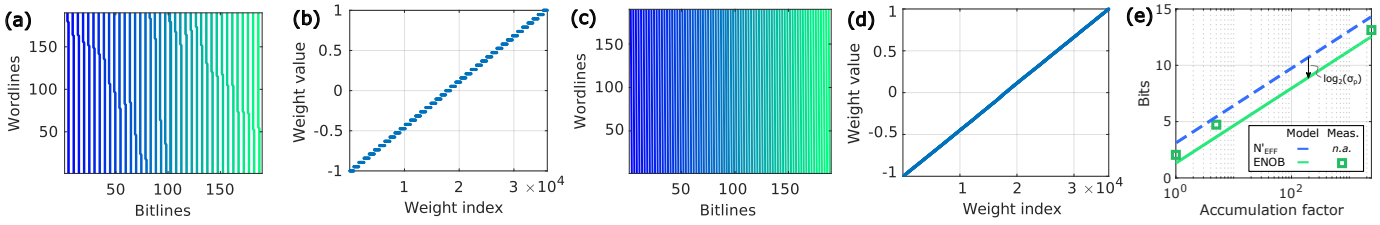

Fig. 5: (a) Contour map and (b) coefficients distribution of the 190×190 programmed coefficients with accumulation $M = 5$. (c) Contour map and (d) coefficients distribution of the 190×190 programmed coefficients with accumulation $M = 2400$. (e) Theoretical (continuous line) and measured (dots) values for ENOB as a function of the accumulation factor M plotted along with the theoretical value of N'_{eff} (dashed line).

A. PCM cells programming

To assess previous method, a set of ∼36K weights was programmed using a differential SET-staircase (SSC) algorithm with a tolerance Δ_p equal to about 15% of the ADC full scale range. A first programming outcome was implemented measuring $z_{i,j}$ with no accumulation ($M = 1$) through the ADCs. Fig. 4(b) reports a contour map of the programmed 190×190-weights matrix, whose normalized distribution is reported in (c) as a function of the programmed weight. Results show that 8 signed levels are programmed within the matrix, so that the effective number of bit is in this case $N'_{\text{eff}} = N_{\text{eff}} = 3.05$.

Before validating the accumulation method, charge samples $z_{i,j}$ variability was firstly characterized as stated in Sec. II-C. To this purpose, a set of 1.6k cells was programmed to different intermediate states by applying random current pulses with no verify step. Then, each cell was read through the ADC to monitor its read noise amplitude $z_{i,j}^{pp}$ (see Fig. 4(c)). Fig. 4(d) shows the distribution of the collected $z_{i,j}^{pp}$, demonstrating that their value is invariably above the ADC quantization step, so that it allows for the precision increase brought by N'_{eff}.

The same amount of MVM weights was then programmed with accumulation $M = 5$. This leads to an estimated $N'_{\text{eff}} = 5.32$ (from (6)). As reported by the experimental results in Fig. 5(a-b), this case allows for the programming of 80 weights (half negative and half positive), which confirms previous estimation of N'_{eff}. An accumulation factor $M = 2400$ was then tested, so that the effective number of bits can be theoretically increased to $N'_{\text{eff}} = 14.14$. The programming outcome is reported in Fig. 5(c-d), where 18049 different signed weights were successfully programmed among the PCM cells array.

Fig. 5(e) summarizes the theoretical N'_{eff} (see (6)) as a function of the accumulation factor M (dashed line). The estimated and measured ENOB are also reported as continuous lines and dots, respectively. The former is calculated replacing $\sigma_\varepsilon = \Delta_p/3$ in (7), whereas the latter is obtained by quantifying the experimental σ_ε.

B. Case study: DNN layers normalization

Despite DNNs can be properly designed to account for their coefficients quantization, the deployment of its layers weights on AiMC cores often necessitates specific rescaling and normalization to enhance overall accuracy [7], [8]. This demands a high-precision programmability across a large number of PCM conductance values to guarantee uniform and accurate performance. This scenario was chosen as validation of the proposed method.

The DNN illustrated in Fig. 6(a), intended for MNIST image classification, was implemented on the testchip utilizing its MVM computation capability. The pixels of the MNIST input images were cropped to 19×20 and encoded in a 7-bit gray scale format. The first layer of the network comprises 380 neurons and employs the ReLU activation function, while the second layer comprises ten neurons and a softmax-operation. The size of the matrix representing the weights of the first and the second layers are 380×380 and 380×10, respectively, involving 380 and 10 BLs of the PCM array. The DNN was initially trained in FP and then quantized in 8-bits signed precision for the coefficients, without employing any specific HW-aware training techniques [9]. The weights were clipped at half the weight distributions to maximize the ADCs output distribution. To maximize the corresponding MVM output range, a rescaling factor was computed for each

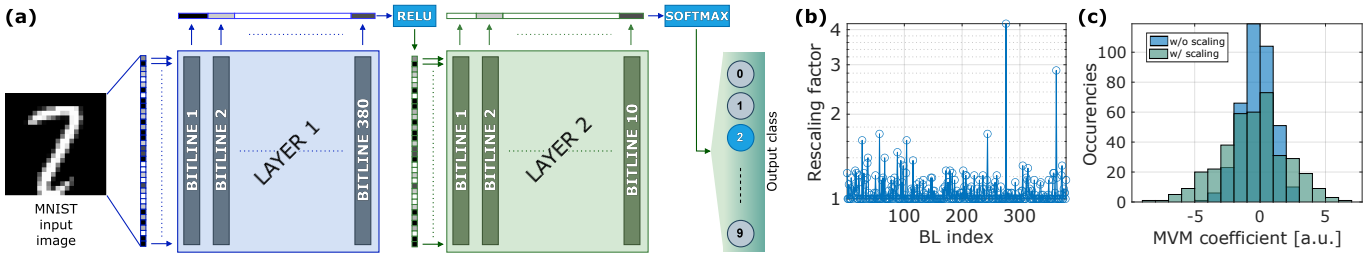

Fig. 6: (a) Schematic representation of the DNN implemented on the PCM AiMC prototype. (b) Rescaling factors used for each BL (log scale). (c) MVM coefficients stored in a sample BL before and after rescaling. The weights numerousness increases due to the BL renormalization.

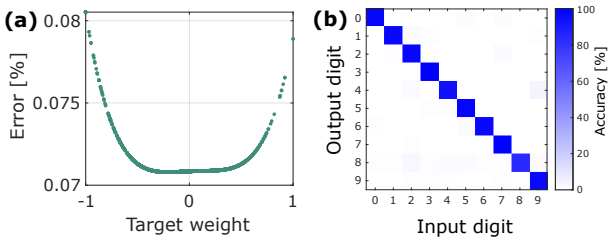

Fig. 7: (a) Relative programming error of the DNN weights. (b) Confusion matrix of the MNIST images classification.

TABLE I
PCM-based AiMC accelerators programming accuracy.

	[3]	[6]	[10]	[11]	**This work**
CMOS techn.	14 nm	90 nm	14 nm	40 nm	**28 nm**
Cells per BL	256	128	512	256	**512**
Program. error	6%	1.95%	3%	*n.a.*	**0.08%**
ENOB	5.5	6	5	8	**10.55**

column (i.e., BL). The two resulting rescaled MVM matrices were programmed into the array. Since each BL requires a specific rescaling factor (as shown in Fig. 6(b-c)), the rescaling increases the numerousness of the coefficients of the stored matrix from 127 (i.e., reduced to 7-bit due to the clipping) to 1901, which require ENOB $\simeq 11$ for accurate representation.

The DNN was then deployed inside the AiMC core targeting $N_{\text{eff}} = 14, 14$ as in Fig. 5(c-d) to grant a sufficient margin for the ENOB (as stated in Sec. II-A). The programming mean relative error (Fig. 7(a)) is $< 0.08\%$, and grants a ENOB $= 10.55$, which is near the 11 bits required to represent the DNN coefficients. The implemented DNN, which exhibits a baseline classification accuracy of 98%, was tested using the MNIST dataset. Computations were performed at room temperature after 12 hours from programming, and the effect of drift was recovered using a Reference Cells Conductance Tracking (RCCT) [5]. The achieved accuracy was 97.5%, and the corresponding confusion matrix is reported in Fig. 7(b).

C. Results discussion

The main results of this work are compared with the State-of-the-Art PCM-based AiMC systems in Table I. The results achieved through the presented analog programming method reach a higher range of stored weights $\hat{z}_{i,j}^{\text{MAX}}$, and the relative programming error $\sigma_\varepsilon / \hat{z}_{i,j}^{\text{MAX}}$ is thus lowered. Despite the largest number of weights per BL, this achievement ensures the highest experimental ENOB. This advantage comes at the cost of longer programming time due to the accumulation, which however is not a severe constraint thanks to the infrequent update of MVM weights. This opens up the possibility for PCM-based AiMC to be employed in further applications which require a high amount values for the MVM coefficients.

IV. Conclusion

This paper presents a programming algorithm of Phase-Change Memory devices for Analog in-Memory Computing workloads. Thanks to this method, it is possible to obtain high-precision, quasi-analog programming of the stored coefficients for Matrix-Vector Multiplications, achieving fairly arbitrary number of conductance states of the PCM cells. The proposed algorithm has been experimentally validated through measurements on a 28-nm FD-SOI STMicroelectronics test chip used to implement a Deep Neural Network classification task. The results demonstrate an improvement of the state-of-the-art weights programming accuracy in terms of an Equivalent Number of Bits (ENOB) of 10.55.

References

[1] N. Verma et al., "In-memory computing: Advances and prospects," *IEEE Solid-State Circuits Magazine*, 2019.

[2] W. Haensch et al., "The next generation of deep learning hardware: Analog computing," *Proc. of the IEEE*, 2019.

[3] R. Khaddam-Aljameh et al., "Hermes-core—a 1.59-tops/mm2 pcm on 14-nm cmos in-memory compute core using 300-ps/lsb linearized cco-based adcs," *IEEE JSSC*, vol. 57, no. 4, pp. 1027–1038, 2022.

[4] R. Vignali et al., "Designing circuits for aimc based on non-volatile memories: A tutorial brief on trade-offs and strategies for adcs and dacs co-design," *IEEE TCAS II: Express Briefs*, 2023.

[5] A. Antolini et al., "A readout scheme for PCM-based analog in-memory computing with drift compensation through reference conductance tracking," *IEEE OJSSCS*, vol. 4, pp. 69–82, 2024.

[6] L. Pistolesi et al., "Differential phase change memory (pcm) cell for drift-compensated in-memory computing," *IEEE Transactions on Electron Devices*, vol. 71, no. 12, pp. 7447–7453, 2024.

[7] M. J. Rasch et al., "Fast and robust analog in-memory deep neural network training," *Nature Communications*, vol. 15, no. 1, p. 7133, 2024.

[8] M. Le Gallo et al., "A 64-core mixed-signal in-memory compute chip based on phase-change memory for deep neural network inference," *Nature Electronics*, vol. 6, no. 9, pp. 680–693, 2023.

[9] V. Joshi et al., "Accurate deep neural network inference using computational phase-change memory," *Nature Communications*, 2020.

[10] P. Narayanan et al., "Fully on-chip mac at 14 nm enabled by accurate row-wise programming of pcm-based weights and parallel vector-transport in duration-format," *IEEE TED*, vol. 68, no. 12, 2021.

[11] W.-S. Khwa et al., "A 40-nm, 2m-cell, 8b-precision, hybrid slc-mlc pcm computing-in-memory macro with 20.5 - 65.0tops/w for tiny-al edge devices," in *2022 IEEE ISSCC*, vol. 65, 2022.