Supplementary Information

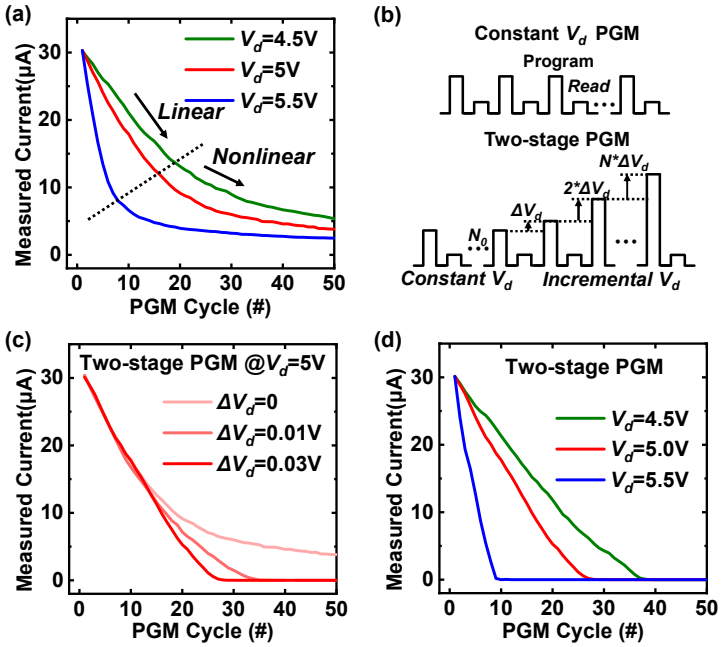

Fig. S1: The proposed two-stage programming scheme. (a) Programming curves (program@ V_g=9 V and read@ V_g=6 V, V_d=0.8 V). (b) The proposed two-stage programming scheme. (c) Programming curves with different V_d incremental steps. (d) Programming curves with excellent linearity by employing the two-stage programming scheme.

In the scenario of working as CAM, Flash states are represented by the read currents, rather than the threshold voltages as in memory mode. The applicable programming scheme should be also different, which demands a linear current modulation during programming. However, when programming under a fixed drain voltage (V_d), the read currents will decrease linearly at first, and then more and more nonlinearly, as shown in **Fig. S1(a)**. This is because as the Flash programming goes on, the electrons trapped in the floating gate is accumulating to saturation, which reduces the programming efficiency [1].

To address this and improve linearity, we propose a two-stage programming scheme, which employs a constant V_d in the first N_0 cycles, and then an increasing V_d with a ΔV which is also incremental, as shown in **Fig. S1(b)**. The principle is utilizing the increasing V_d to compensate for the influence of electron saturation. The effects of different V_d incremental steps with initial V_d=5 V are shown in **Fig. S1(c)**. As the step increases, we can see that the programming linearity is recovered gradually. Besides, by selecting the step sizes carefully, we can obtain an excellent linearity for different initial V_d, as shown in **Fig. S1(d)**. Hence, we can obtain different programming speeds and current steps by selecting the different initial V_d and corresponding ΔV.

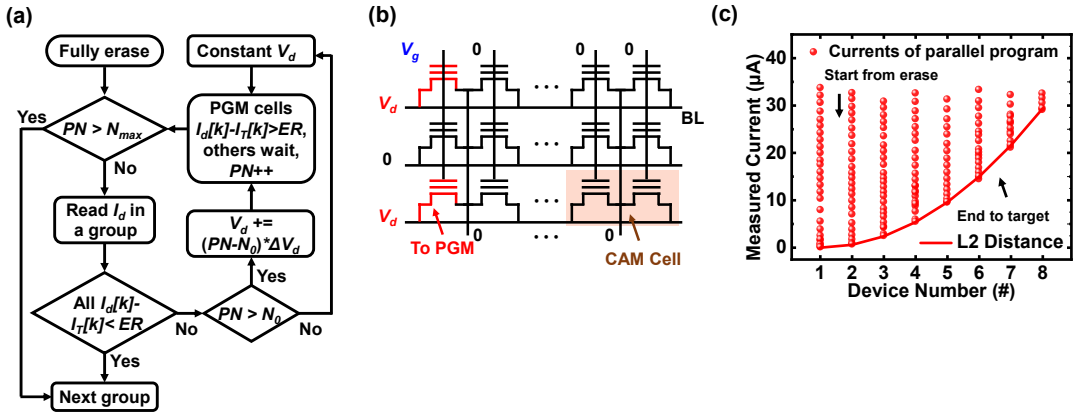

Fig. S2: The employed parallel programming scheme. (a) Flow chart of the parallel scheme. (b) Array structure and parallel programming operation. (c) Programming results in a parallel group of 8.

In addition to improving the programming linearity of single transistor, parallel programming is another effective approach to speed up on-the-fly learning. The flow chart of the employed parallel programming scheme is shown in **Fig. S2(a)**, which is combined with the aforementioned two-stage programming scheme. The parallelism is along with different BLs, where the cells needed to be programmed are applied with V_d, and others connected to ground, as shown in **Fig. S2(b)**. According to the erasing mechanism, the Flash transistors in the same WL will be erased together. Therefore, they should be erased first, and then programmed to the respective states for termination, which guarantees no erasing operation during programming. **Fig. S2(c)** shows the whole process of parallel programming in a group of 8. We can see that the Flash currents start from the erased states and gradually decrease during the programming operation. When one Flash transistor reaches its target, the corresponding programming is terminated.

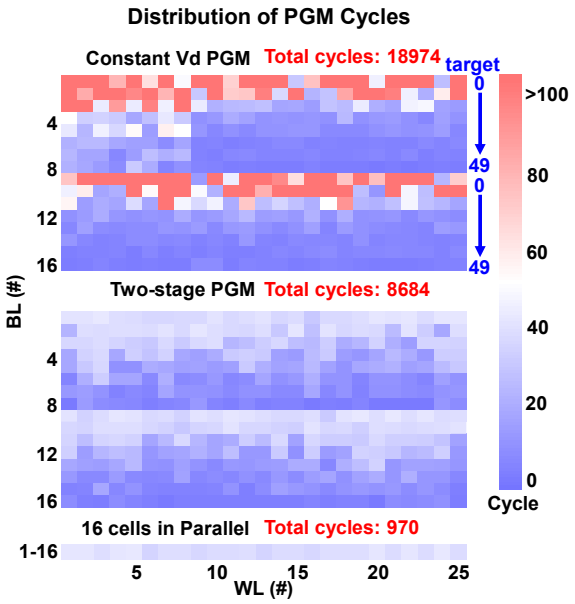

Fig. S3: The comparison of programming cycles in three programming schemes. PGM cycles are dramatically reduced by parallel scheme. The target conductance is set by the L2 distance in each column for fair comparison. (measured @400cells)

Fig. S3 shows the comparison of required programming cycles in three programming schemes, constant V_d programming, two-stage programming alone, parallel and two-stage programming together, respectively. Noted that the number of cycles required in parallel scheme depends on the smallest target currents in the group, therefore, for a fair comparison, the targets in each group are set to cover all the states of 3-bit L2 distance. We set the parallelism as 16 (BL number), and the target currents in each group are 0, 1, 4, 9, 16, 25, 36, 49, 0, 1, 4, 9, 16, 25, 36, 49. By jointly employing the parallel and two-stage programming schemes, we can get a 19.6-fold (average of 25 WLs) reduction in total programming cycles, which provides the basis for rapid and precise on-device lifelong learning.

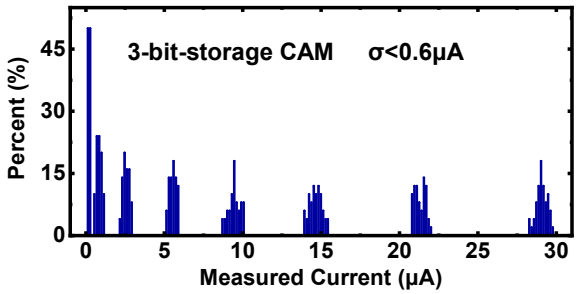

Fig. S4: The overall current distribution of 3-bit-storage CAM. This degree of current variation (σ<0.6 μA) is proved to have almost no degradation to accuracy (<0.15%) in **Fig. 4(d)** of main text.

1

2

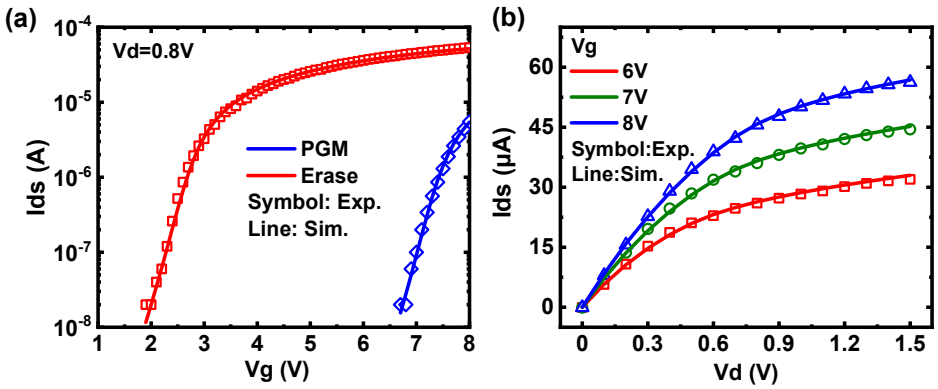

Fig. S5: The fitted (a) I_d-V_g and (b) I_d-V_d curves of NOR Flash transistors based on BSIM3v3 model calibrated with experimental data. The model shows a good consistency with the experiments. The modified parameters include the thickness of oxide layer, saturation velocity of electron, carrier mobility, and source-drain series resistance.

1

2

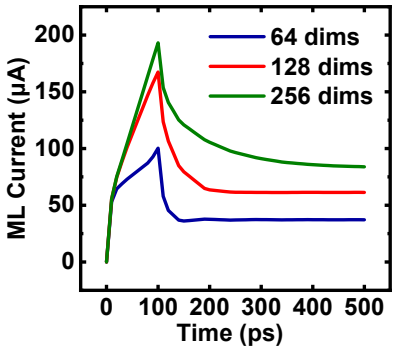

Fig. S6: Search latency with different word widths in the proposed CAM design. The search latency will increase as the word width increases due to the increasing parasitic resistance and capacitance in MLs. We can see that the search latency in 256-dimension word is almost 500 ps, which increases by 3.3-times compared to 64-dimension word. Therefore, multi-bit capability in CAM is also crucial to reduce the word width (related to the circuit area and search energy) and search latency.

1

2

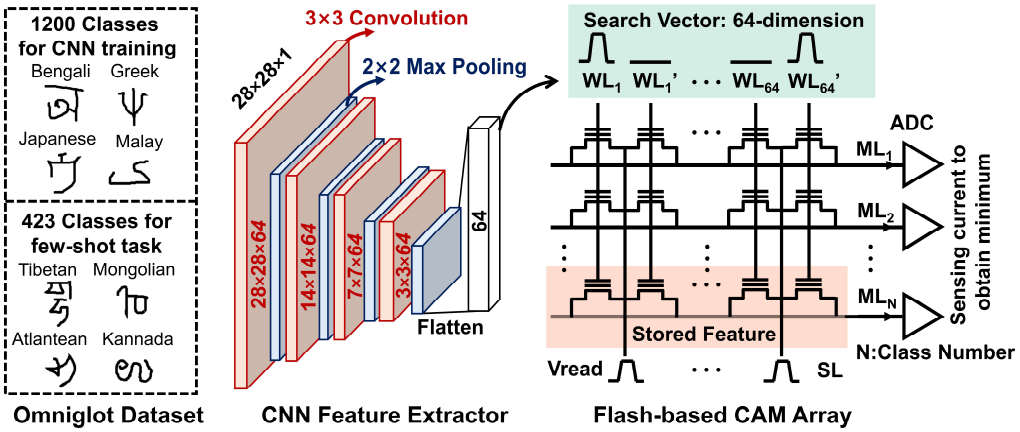

Fig. S7: Experiments for few-shot learning. Pre-trained 4-layer CNN is used to extract features. During few-shot learning, features of support set are written into the Flash-based CAM. During inference, search vector of query is compared with all the stored contents by calculating pair-wise L2 distance. Predicted label is given by sensing the minimum current of MLs.

The schematic of our experiments is shown in **Fig. S7**. we choose the widely used Omniglot dataset for demonstration. In this dataset, there are 1623 handwritten characters (classes) from different alphabets, and each character contains 20 examples from different people. We randomly choose 1200 classes for meta-training of feature extractor and the rest of 423 classes for configuring few-shot learning tasks. We employ a four-layer CNN as the feature extractor, which consists of 4 convolutional modules, each including 64-channel 3×3 kernels, batch normalization, ReLU, and 2×2 max-pooling. The output of final layer is flatted to a 64-dimension vector.

During meta-training, for one n-way k-shot task, we randomly sample n different classes (each class with $k + x$ examples) from 1200 classes to establish support set (k examples) and query set (x examples). Here, both the support set and query set are labeled examples. The CNN is then trained to minimize the error of predicting labels in the query set by the back-propagation (BP) algorithm, which is called an "episode". By repeating this process (randomly sampling few-shot tasks and BP training), the CNN is gradually meta-trained to be capable of employing a general scheme to map the input data to their appropriate labels regardless of their specific contents, namely learning to learn [2]. Note that the CNN weight parameters are fixed after meta-training and will no longer change during the following phase. During few-shot learning, the 64-dimension features of support set are first extracted by the CNN, quantized to 3-bit according to the scheme of L2 distance, and then stored in the Flash-based CAM. During inference, the extracted query features are coded and transformed into a series of WL voltages for computing L2 distance with all the stored features in parallel. The predicted label is given by sensing MLs for minimum current.

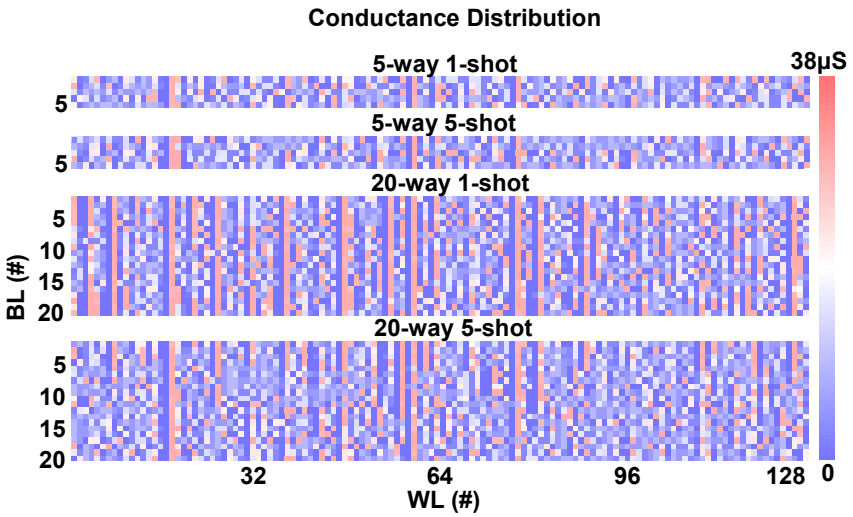

Fig. S8: Conductance distribution of the stored 64-dimension feature vectors in different few-shot learning tasks. The features of 5-shot task are average results across different shots.

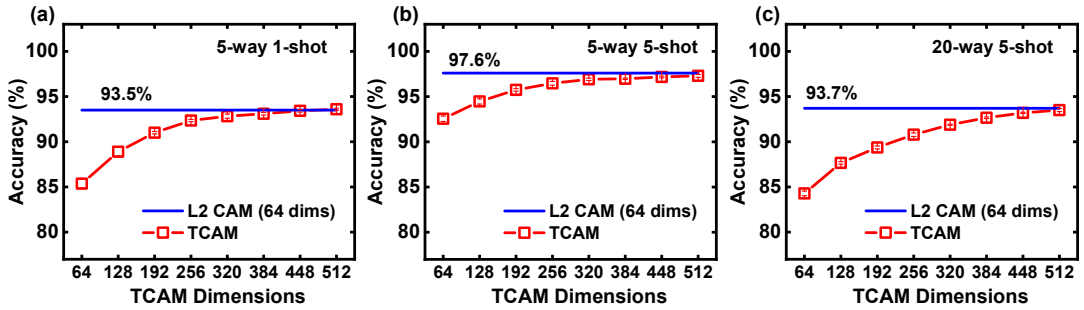

Fig. S9: To achieve the same accuracy with the proposed L2 CAM, the word width of TCAM needs to be extended.

1

2

3

4

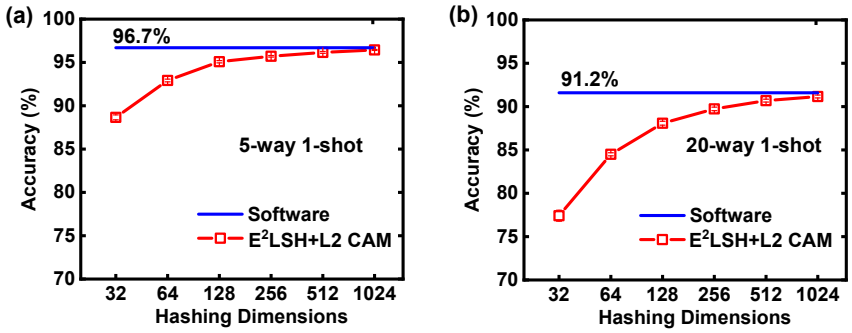

Fig. S10: Increase L2 CAM dimensions by using p-stable locality sensitive hash algorithm (E^2LSH) for higher inference accuracy.

The LSH algorithm is to solve the problem of approximate nearest neighbor search. The original LSH method embeds the original feature-vector space into the Hamming space and converts the distance measure of original space to the Hamming distance, which performs bitwise XOR operation in two equal-length binary sequences and accumulates the results of "1" [3]. It has been applied in TCAM, but is difficult to directly extend to the multi-bit CAM.

To address this and apply the LSH method to our proposed multi-bit-storage L2 CAM, we employ the p-stable LSH method, which can directly perform the locality-sensitive hash operation in Euclidean space and is also called E^2LSH [4]. The hashing function $h_{a,b}(v)$ of p-stable LSH is defined in equation (1).

$$h_{a,b}(v) = \left\lfloor \frac{a \cdot v + b}{r} \right\rfloor \tag{1}$$

The parameter a is a random vector that has the same dimension as the feature vector v. Each element in a is randomly and independently generated from a p-stable distribution (p=2 is Gaussian distribution). The parameter b is a random number in the range of [0, r], of which r is an empirical hyperparameter. Besides, the final result needs a round-down operation. One hashing function acting on the original feature vector can generate one value for the new mapping vector. Therefore, by establishing more hashing functions, we can map the 64-dimension feature vector to a larger width and increase the inference accuracy of MANN. as shown in **Fig. S10**.

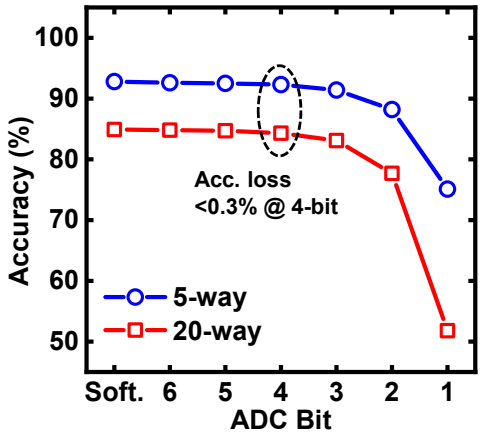

Fig. S11: The impacts of ADC quantization precision on the inference accuracy. 4-bit is enough for the quantization precision of ML sum current, with an accuracy loss less than 0.3% for both of 5-way and 20-way tasks.

1

2

3

Supplementary References

1. Xiang, Y. C., Huang, P., Yang, H. Z., Wang, K. L., Han, R. Z., Shen, W. S., Kang, J. F., et al. (2019). Storage reliability of multi-bit flash oriented to deep neural network. In 2019 IEEE International Electron Devices Meeting (IEDM) (pp. 38-2). IEEE.
2. Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. (2016). Meta-learning with memory-augmented neural networks. In International conference on machine learning (pp. 1842-1850). PMLR.
3. Andoni, A., and Indyk, P. (2008). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. Communications of the ACM, 51(1), 117-122.
4. Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. S. (2004). Locality-sensitive hashing scheme based on p-stable distributions. In Proceedings of the twentieth annual symposium on Computational geometry (pp. 253-262).