

# A Self-Supervised Neuromorphic Processor Using High-Dimensional Representations for Cognitive Map Navigation

Anqin Xiao, Luyu Yang, Yuhan He, Hengtian Zhang, Ziyi Yang, Lirong Zheng, Zhuo Zou  
*College of Future Information Technology, Fudan University, Shanghai, China*  
 Email: aqxiao22@m.fudan.edu.cn, zhuo@fudan.edu.cn

**Abstract**—This work proposes a self-supervised neuromorphic processor using high-dimensional representations for cognitive map navigation. By employing the Cognitive Map Learner (CML), it enables agents to explore and understand diverse environments online through random walks. To enhance path planning, the agent’s actions and observations are embedded into high-dimensional state spaces. This embedding creates a sense of direction, simplifying navigation into a retrieval process within an Associative Memory (AM). We design an energy-efficient processor that features a scalable multi-core hardware architecture with precision flexibility, combined with an on-chip random walk training engine. To balance the precision of the model with hardware overhead, two hardware-software co-design strategies are proposed. The first is a Content-Addressable Memory (CAM)-based approach for AM access, which reduces the number of memory access by up to 25%. The second involves high-dimensional matrix sparsity optimizations, reducing computation operations to less than 8%. We simulate this processor by a 40-nm CMOS technology, which has 2.88 mm<sup>2</sup> core area with 15.8 mW power at a frequency of 140 MHz. Compared to previous processors, our experiments show that the proposed processor achieves outstanding success rates of 99.9%, 96%, and 98.7% on 100 2D nodes, 125 3D nodes, and 25 abstract map nodes with obstacles, respectively. In terms of energy efficiency, it delivers a path planning result of 28 nJ/node and 35 nJ/node in 2D and 3D maps, offering a 1.2x to 2.9x improvement over the state-of-the-art.

**Index Terms**—Cognitive Map Learner (CML), navigation, neuromorphic processor, high-dimensional representation, self-supervised learning

## I. INTRODUCTION

Intelligent path planning and navigation are becoming increasingly critical for resource-constrained agents and robots, especially with the rapid advances in Machine Learning (ML) [1], [2], [3]. These systems must not only operate efficiently within limited resource budgets, but also demonstrate adaptability to dynamic environments. Such capabilities enable their effective deployment in diverse applications, including indoor warehouse automation and transportation network models.

Traditional algorithms predominantly rely on geometric computations, such as the Dijkstra algorithm and A\* [4], to address spatial navigation challenges. Although these methods are effective in static environments, they often face significant challenges in adapting to complex dynamic environments. In particular, they struggle to adjust instantaneously to changes

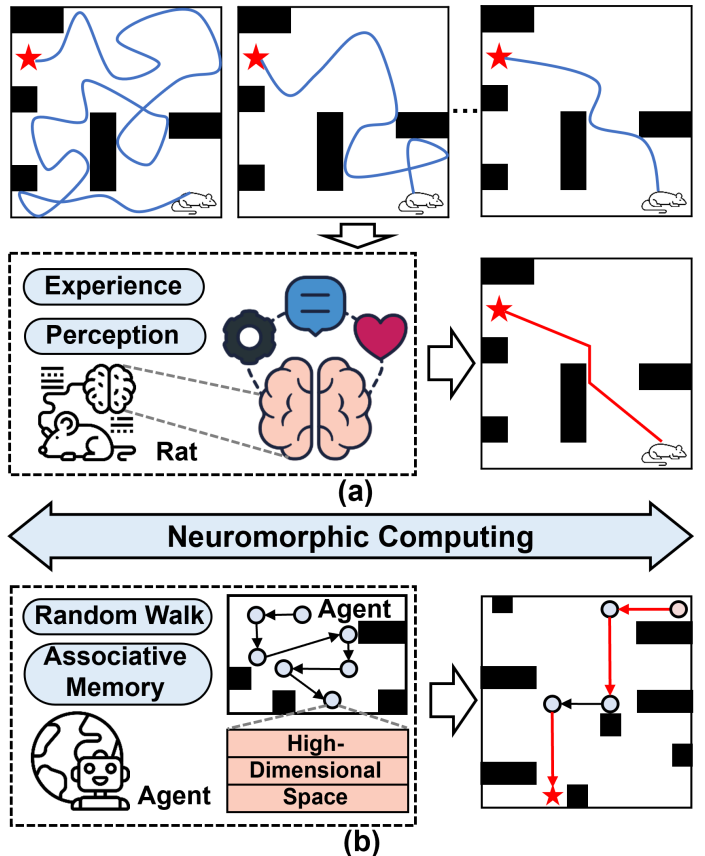


Fig. 1. (a) Brain leverages experience and perception for navigation. (b) Heuristic neuromorphic computing for navigation.

in goals or to handle contingencies that render certain actions unavailable. As a result, when the start or the goal changes, these algorithms must restart the navigation from scratch, leading to intensive computation. Although some ML-based methods enable agents to interact with the environment and process data efficiently [5], [6], [7], [8], their success rate still falls short compared to geometric methods. In addition, energy efficiency remains a challenge for edge applications.

As illustrated in Fig. 1(a), the brain may leverage experience and perception for navigation [9], rather than performing geometric computations of distance and direction. Inspired by this heuristic process, the Cognitive Map Learner (CML)

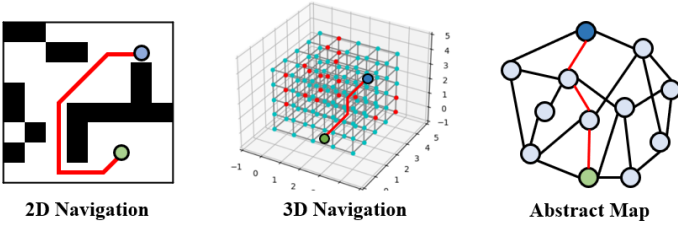


Fig. 2. The various applications supported by this processor.

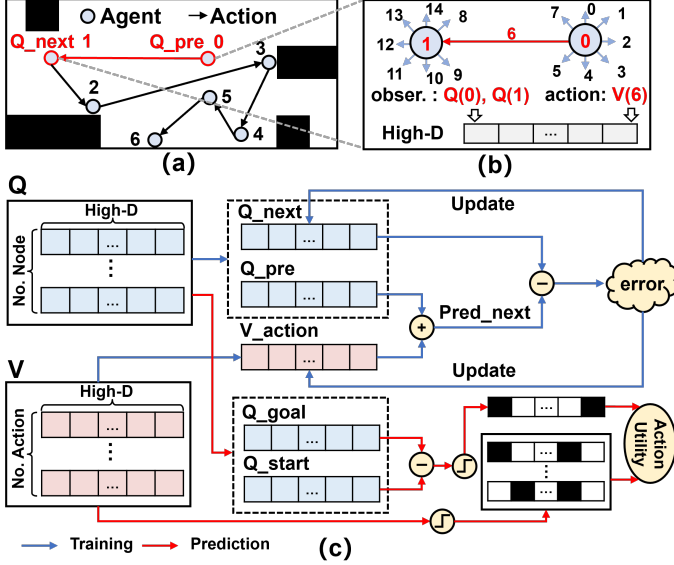


Fig. 3. (a) Path planning in 2D grid map with obstacles. (b) Using high-dimensional vectors by embedded observations and actions of agent. (c) CML computing flow of training and prediction.

[10], [11] leverages online learning to emulate cognitive mechanisms, selecting an action or sequence of actions based on the desirability of the agent's observations to determine the shortest path. As shown in Fig. 1(b), the self-supervised CML shares with Transformers [12] the characteristic that it does not require a teacher for backpropagation learning. Additionally, the outcome of its learning process is the embedding of external tokens, such as an agent's actions and observations, into a high-dimensional representation. As a result, it enables efficient computation and supports on-chip random walk training. Furthermore, the high-dimensional vectors facilitate parallel processing and ensure low latency.

Building on the CML, our work proposes an energy-efficient neuromorphic processor designed with hardware-software co-design strategies. The processor features a scalable multi-core hardware architecture with precision flexibility, combined with an on-chip random walk training engine. To balance model precision with hardware overhead, hardware-software co-design strategies are proposed. These include a Content-Addressable Memory (CAM)-based approach, which reduces memory accesses by up to 25%, and high-dimensional matrix sparsity techniques, which decrease the number of computational operations to less than 8%. Simulated in 40-nm CMOS technology, the processor achieves a core area of 2.88 mm<sup>2</sup>,

## Algorithm 1 Navigation with Q and V Matrices (CML-based)

### Parameters:

- $N_{\text{node}}$ : Number of nodes
- $N_{\text{act}}$ : Number of actions
- $D$ : Embedding dimension
- $n_{ts}$ : Steps per trajectory
- $n_{\text{step}}$ : Number of trajectories

Initialize  $Q[N_{\text{node}}][D]$ ,  $V[N_{\text{node}} \times N_{\text{act}}][D]$  randomly

---

TRAINING PHASE

$n = \mathcal{U}(0, N_{\text{node}} - 1) \leftarrow \text{start\_position}$

$Q_n = Q[n][\ ]$

**for**  $i = 1$  to  $n_{\text{step}}$  **do**

**for**  $j = 1$  to  $n_{ts}$  **do**

$a = \mathcal{U}(0, N_{\text{act}} - 1) \rightarrow V_a = V[a][\ ]$

$n' = \text{step}(n, a) \rightarrow Q_{n+1} = Q[n'][\ ]$

$\hat{Q}_{n+1} = Q_n + V_a \rightarrow e = \|Q_{n+1} - \hat{Q}_{n+1}\|$

    Update  $Q[n][\ ]$  and  $V[a][\ ]$  using error  $e$  with learning rates  $\alpha$  and  $\beta$  respectively

$n \leftarrow n' \{ \text{Update current position} \}$

**end for**

**end for**

---

PREDICTION PHASE

$s \leftarrow \text{start\_position}$ ,  $g \leftarrow \text{goal\_position}$

$\mathcal{A}(s) \leftarrow \{a_1, \dots, a_k\}$

$Q_s \leftarrow Q[s][\ ]$ ,  $Q_g \leftarrow Q[g][\ ]$

**while**  $s \neq g$  **do**

$V_{\text{binary}} \leftarrow \text{Binarize}(V)$

$a^* \leftarrow \arg \max_{a \in \mathcal{A}(s)} ((Q_g - Q_s)^\top V_{\text{binary}}[a][\ ])$

$s \leftarrow \text{step}(s, a^*)$

$Q_s \leftarrow Q[s][\ ]$

**end while**

---

operates at 15.8 mW at 140 MHz. In our experiments, it achieves success rates of 99.9%, 96%, and 98.7% across 100 2D nodes, 125 3D nodes, and 25 abstract map nodes in Fig. 2, respectively. And it demonstrates superior energy efficiency of 28 nJ/node and 35 nJ/node in 2D and 3D maps, offering a 1.2x to 2.9x improvement over state-of-the-art solutions.

## II. ALGORITHM AND FRAMEWORK

### A. Cognitive Map Learner

As illustrated in Fig. 3(a), observations represent the agent's state (identified by an ID) at a specific moment, while actions correspond to the operations that lead to the next state. For instance, in a path planning task, observations correspond to the agent's location on the node map, whereas actions represent the possible directions for movement.

In the CML algorithm, the number of observations is determined by the size of the node map. For each observation, the number of available actions depends on the possible directions, with obstacles restricting certain actions. Both observations and actions are embedded using a linear function into a high-dimensional continuous space, referred to as  $Q$  and  $V$ , as depicted in Fig. 3(b). The computing flows for both training

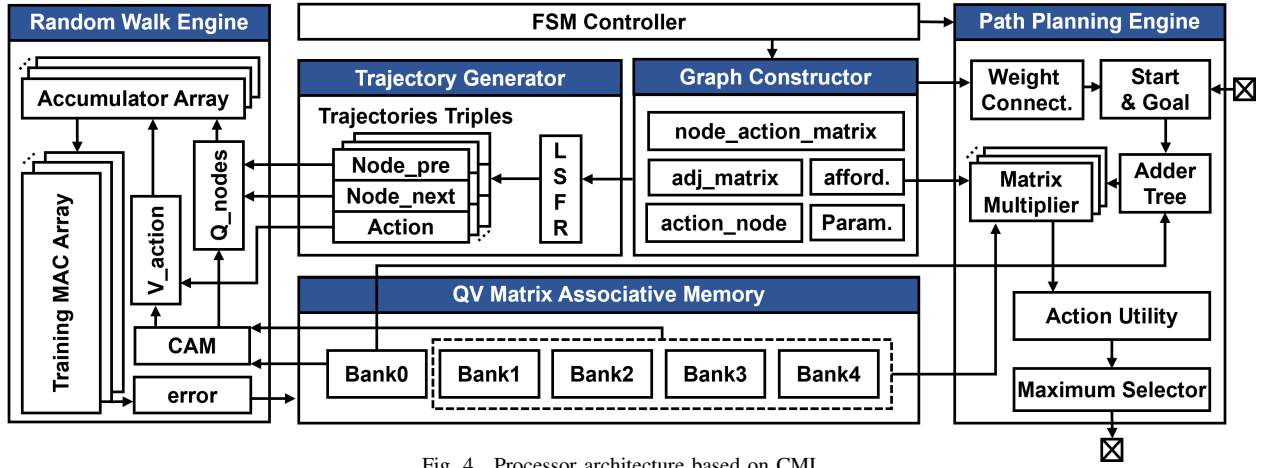


Fig. 4. Processor architecture based on CML.

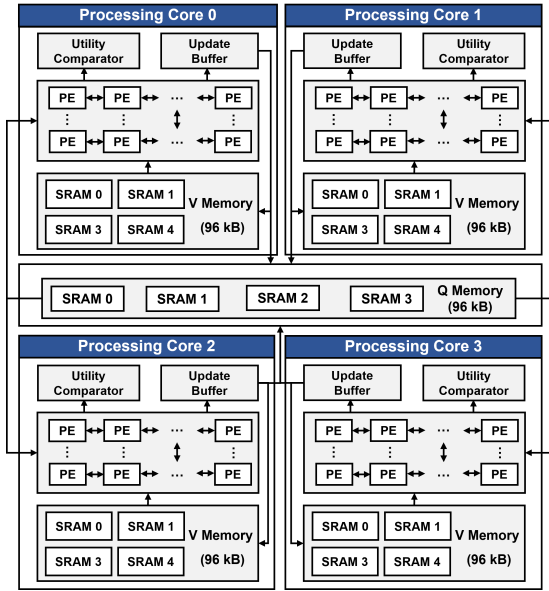


Fig. 5. Scalable multi-core structure for processing QV matrices.

and prediction are illustrated in Fig. 3(c) and are described in detail in the following section.

### B. Self-Supervised Random Walking

The training process operates without the need for back-propagation under teacher supervision, relying on random walks to reduce learning costs. Self-supervised learning facilitates efficient online learning, where plasticity rules are applied during the walking phase and are further reinforced during the replay of episodic memories in subsequent offline prediction phases.

The specific training process is shown in Algorithm 1 (Training phase). Specifically, the training process utilizes trajectories consisting of  $n_{step}$  iterations, where each trajectory has a length of  $n_{ts}$ . For a randomly given trajectory, the process begins at a starting node  $n$ , followed by selecting a next node  $n'$  from a set of nodes. The high-dimensional

vectors  $Q_n$  and  $Q_{n+1}$  are retrieved from the matrix  $Q$  using these nodes as their IDs.

An action  $a$ , defined between  $n$  and  $n'$ , is chosen from the action space, retrieving its associated vector  $V_a$  from the matrix  $V$ . The state of the predicted next node  $\hat{Q}_{n+1}$  is then computed by adding  $V_a$  to  $Q_n$ . The prediction error  $e$ , calculated as the difference between  $Q_{n+1}$  and  $\hat{Q}_{n+1}$ , is used to update the matrices  $Q$  and  $V$  with different learning rates. Finally, the current node  $n$  is updated to  $n'$ . This process is repeated iteratively until a trajectory of length  $n_{ts}$  is completed.

### C. Utility Estimation for Navigation

The detailed prediction process is described in Algorithm 1 (Prediction Phase). It enables efficient computational by estimating the maximum utility of a set of actions. These utility estimates are similar to value estimates in reinforcement learning, but do not depend on a policy. Furthermore, it encodes information from  $\langle node, action, next\_node \rangle$  triples into a static cognitive map. This static representation eliminates the need for repeated real-time querying and processing of large amounts of historical experience data (such as past trajectories and outcomes).

This process employs iterative triples to navigate from the start position  $s$  to the goal position  $g$ . The current action is selected based on the maximum utility. The positions  $s$  and  $g$  are derived from the matrix  $Q$ , which provides the high-dimensional vectors  $Q_s$  and  $Q_g$ . To enhance computational efficiency with small precision loss, the  $V$  matrix is binarized into  $V_{binary}$ , referred to as  $V_b$ . The high-dimensional vector  $V_b[a]$ , associated with action  $a$ , is evaluated through the scalar product with  $(Q_g - Q_s)^T$ , determining the utility of action  $a$ . The action with the highest utility among the available options is then selected.

## III. NEUROMORPHIC CML PROCESSOR

To address the edge resource-constrained path planning task using the CML model, we have designed a parallel multi-core neuromorphic processor that enables energy-efficient on-

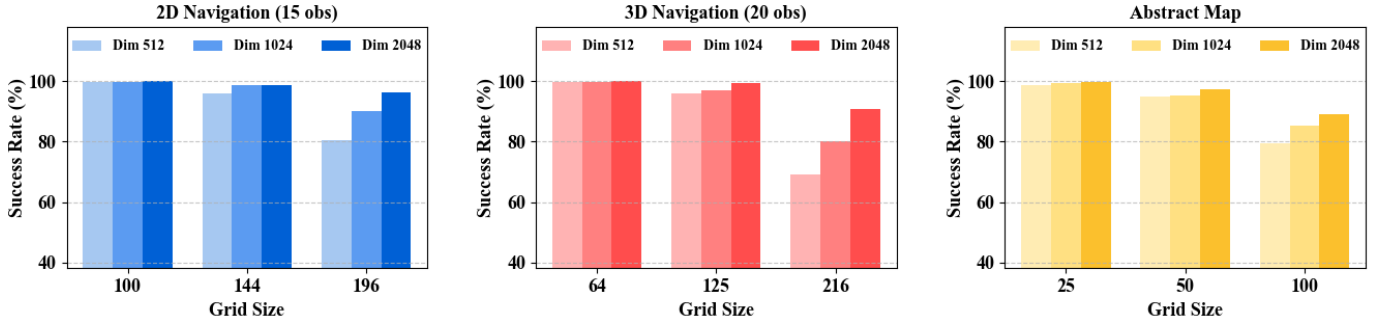


Fig. 6. Precision flexibility with different dimensions with multi-core structure.

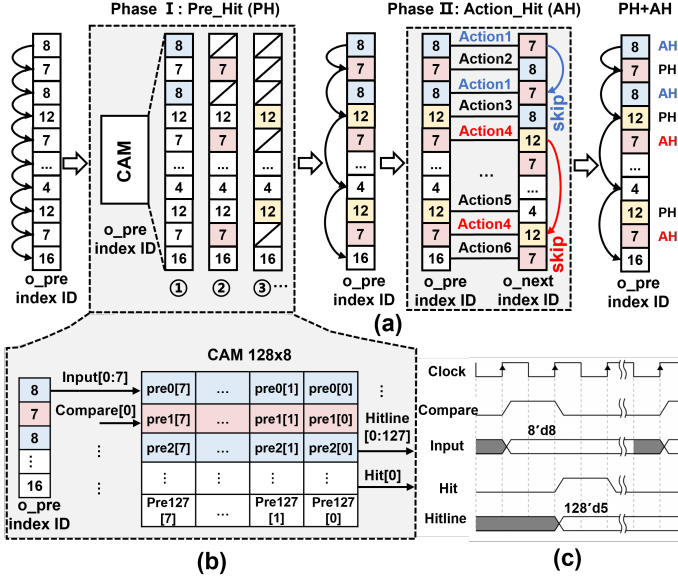


Fig. 7. (a) CAM-based memory access optimization in two phases. (b) The CAM specification and (c) timing diagram.

chip learning and predictive computing. Additionally, it supports flexible multi-precision operations by scaling the high-dimensional vectors of  $QV$  matrices. The detailed hardware architecture is presented in the following sections.

### A. Overall Hardware Architecture

As shown in Fig. 4, the proposed CML processor is composed of five modules: (a) **Random Walk Engine**: Processes random trajectories to retrieve the  $QV$  vectors from associative memories (memory access optimized by Content-Addressable Memory) and outputs the updated  $QV$  vectors by training error. (b) **Path Planning Engine**: Accepts map information, including the relationships between nodes and actions, as well as external start-goal navigation nodes. It then computes the action utilities and outputs the path-planning results. (c)  **$QV$  Matrix Associative Memory**: Stores the embedded high-dimensional  $QV$  state space, implemented using SRAM, which consists of one memory bank for the  $Q$  matrix and four memory banks for the  $V$  matrix. (d) **Trajectory Generator**: Generates random trajectory triples using a Linear-Feedback

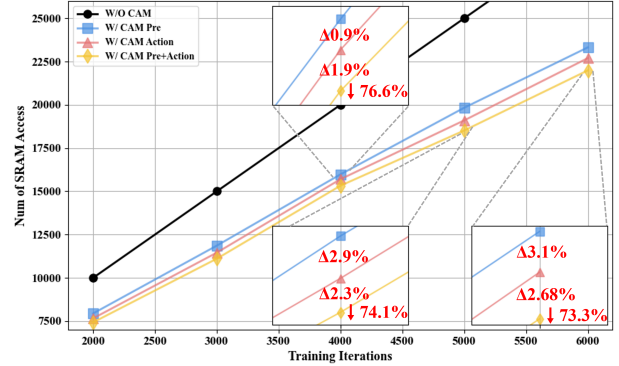


Fig. 8. The results of reducing memory access through two-phase CAM utilization. (Tested on 5x5x5 3D grid map with 20 obstacles)

Shift Register (LFSR). (e) **Graph Constructor**: Stores map information in a graph structure along with global parameters.

This architecture features 480 kB of SRAM and 0.125 kB of CAM. The SRAM is divided into 96 kB for the matrix  $Q$  and 384 kB for the matrix  $V$ . As a result, it supports up to 128 observations and 512 actions within a 512-dimensional high-dimensional vector for an agent. During the computation process, the  $Q$  and  $V$  matrices are quantized using INT-12 precision.

### B. Scalable Multi-Core Structure with Precision Flexibility

The scalable multi-core hardware design enables precision flexibility by supporting  $QV$  matrices with 512, 1024, and 2048 dimensions. When processing higher-dimensional vectors, we can achieve greater navigation accuracy. However, this comes at the cost of higher latency. To address this, we adopt a scalable multi-core processing approach. In environments with simpler maps and lower accuracy requirements, we utilize a single-core serial processing scheme, which reduces static power consumption by turning off unused SRAM blocks while maintaining acceptable latency.

As shown in Fig. 5, the  $Q$  matrix is mapped to a single core, with each  $Q$  vector accessed by four parallel SRAMs. In contrast, the  $V$  matrix can be distributed across up to four parallel processing cores, as the latency-constrained prediction process requires more  $V$  vectors (actions) compared to  $Q$  vectors (observations).

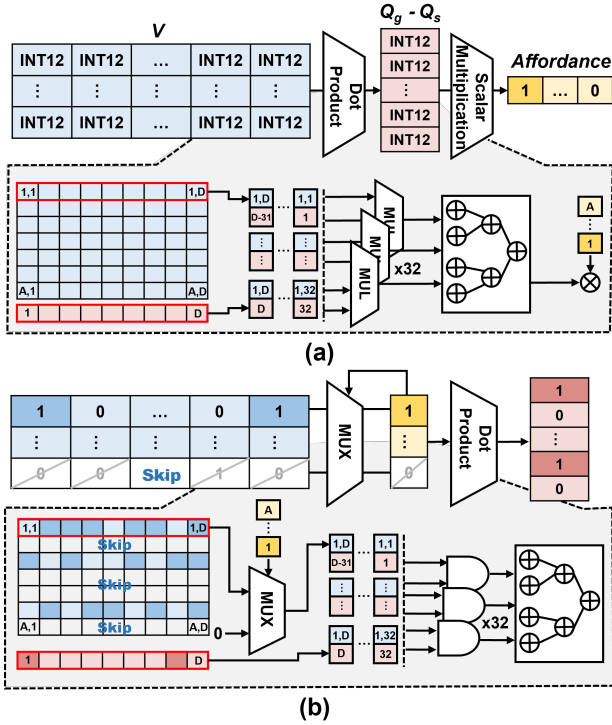


Fig. 9. (a) Flow of Software computation and hardware implementation. (b) Hardware and software co-design of prior implementation.

As shown in Fig. 6, experiments conducted on 2D, 3D and abstract maps demonstrate that small node maps achieve high success rates using a 512-dimensional single-core configuration, offering low energy consumption and acceptable latency. For larger node maps, 1024- or 2048-dimensional multi-core configurations are more effective, delivering higher success rates with reduced latency.

### C. Hardware-Software Co-Design

In this work, our aim is to balance the precision of CML model and hardware overhead through a hardware-software co-design approach. Specifically, we propose a CAM-based  $QV$  memory access scheme to minimize the latency and energy consumption of the on-chip random walk, along with a sparsity-aware high-dimensional vector computation strategy that reduces the number of operations while enhancing energy efficiency.

1) *Accelerating with Content-Addressable Memory*: During the on-chip random walking process, the CML utilizes iterative triples  $\langle pre\_node, action, next\_node \rangle$  to compute the training error, as illustrated in Algorithm 1. The training error is expressed as  $Q_{n+1} - (Q_n + V_a)$ . As shown in this equation, five memory access operations are required: three SRAM reads and two SRAM writes for  $Q_n$  and  $V_a$ . Consequently, a large number of triples leads to increased memory access, quantified by the following equation:

$$\text{No. access times} = n_{ts} \times 5 \quad (1)$$

The  $n_{ts}$  means the length of a trajectory, which also means the number of triples.

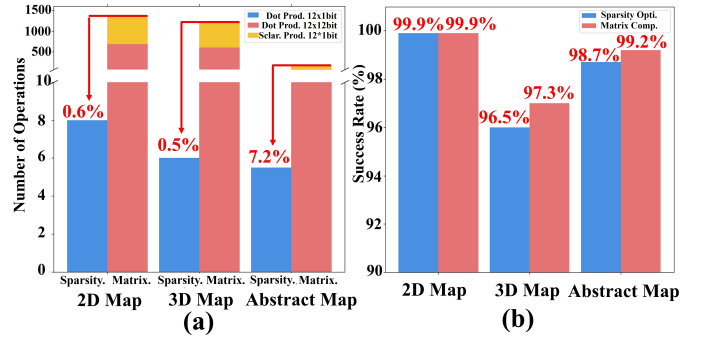


Fig. 10. Leveraging the sparsity of  $QV$  matrices to optimize (a) the number of computational operations while maintaining (b) acceptable precision.

TABLE I  
COMPARISON WITH PRIOR WORKS ON NAVIGATION PRECISION IN A 2D 10x10 GRID MAP.

	Success Rate	Direction	Model
ICRA'22 [5]	89.20%	4	NN-HDC
DATE'22 [7]	95%	4	FRL
ICRA'24 [8]	95%	4	HDC-RL
This Work	98.7%	4	CML
	99.9%	8	CML

As shown in Fig. 7(a), the memory indices of  $pre\_nodes$  for each trajectory are stored in a CAM, which performs a comparison operation to search for entries with matching content. In Fig. 7(b), a 128x8 CAM is utilized in our processor. It searches for 8-bit input data and outputs the addresses of the matched content in a 128-bit hitline. Specifically, the memory index of  $pre\_nodes$  is processed in two phases. In Phase *I*, all  $pre\_nodes$  with the same index ID are identified, allowing the  $Q$  vector to be accessed once without multiple accesses. In Phase *II*, a subset of these  $pre\_nodes$  is selected based on matching next nodes, enabling both  $Q$  and  $V$  vectors to be accessed simultaneously. The optimization result is quantified by the following equation:

$$\text{No. access times} = \text{Num}_{pre} \times 4 + 5 + \text{Num}_{nohit} \times 5 \quad (2)$$

where  $Num_{pre}$  represents the number of matched  $pre\_nodes$ , and  $Num_{nohit}$  denotes the number of unmatched  $pre\_nodes$ . As shown in Fig. 8, the number of SRAM access is reduced to 73.3% with 6000 training iterations, compared to the scenario without CAM optimization.

2) *Sparsity-Aware High-Dimensional Vector Computation*: During the path prediction process, the computation of action utilities accounts for the majority of the operational overhead. As illustrated in Algorithm 1, the utilities are represented as a series of matrix and vector operations, expressed by the equation  $(A, D) \times (D, 1) * (A, 1)$ , where  $A$  denotes the number of actions and  $D$  represents the dimensionality of the vector. As shown in Fig. 9(a), an INT-12bit dot product is performed between  $V$  and  $(Q_g - Q_s)$ , followed by a scalar multiplication with a 1-bit affordance vector. Here,  $V$  represents the  $V$

TABLE II  
COMPARISON WITH PREVIOUSLY IMPLEMENTED PATH PLANNING PROCESSORS.

	TCAS-I'18 [13]	JSSC'21 [14]	VLSI'23 [15]	This Work
<b>Technology</b>	65 nm	40 nm	28 nm	40 nm
<b>Planning Algorithm</b>	RL	RRT	GNN	CML
<b>Die Area</b>	16 mm <sup>2</sup>	3.6 mm <sup>2</sup>	5.1 mm <sup>2</sup>	4 mm <sup>2</sup>
<b>Clock Frequency</b>	7 MHz	200 MHz	200 MHz	140MHz
<b>On-Chip Learning</b>	No	No	No	Yes
<b>Prediction Energy Efficiency</b>	2D: 79 nJ/node	2D: 78 nJ/node 3D: 103 nJ/node	3D: 41 nJ/node	2D: 28 nJ/node 3D: 35 nJ/node

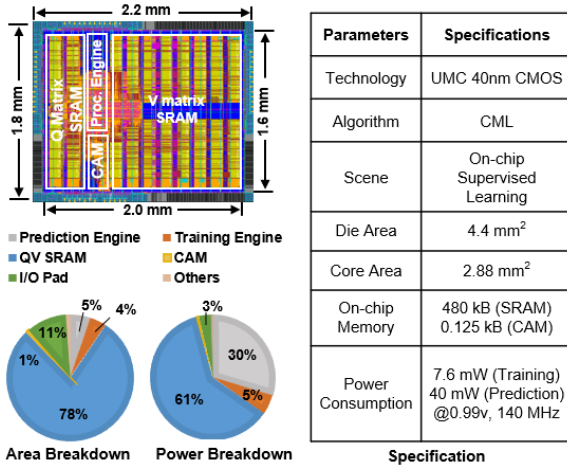


Fig. 11. Layout, specification and breakdown of simulated processor.

matrix,  $Q_g$  is the  $Q$  vector of the goal node,  $Q_s$  is the  $Q$  vector of the start node, and the affordance vector indicates the available actions at the current position. However, the hardware implementation of this equation involves computing a large matrix of size  $A \times D$ , which consumes excessive multipliers in INT-12 precision.

As illustrated in Fig. 9(b), the sparsity of the  $V$  matrix is leveraged to reduce both the number of operations and the complexity of matrix computations. Specifically, the affordance vector is used as a multiplexer (MUX) for the  $V$  matrix, skipping rows of the matrix corresponding to zero values in the affordance vector. Additionally, the INT-12bit precision is binarized to 1-bit precision, significantly lowering the complexity of the dot-product computation while maintaining an acceptable precision loss. As shown in Fig. 10, the sparsity matrix optimization requires less than 8% of the computational operations, with a precision loss of less than 1%.

#### IV. IMPLEMENTATION AND EVALUATION

##### A. Hardware Simulation Results

The CML-based processor is synthesized in a 40 nm CMOS process, with a layout, specification, and breakdown shown in Fig. 11. It occupies an area of 4.4 mm<sup>2</sup>, including IO pads, while the core area accounts for 2.88 mm<sup>2</sup>. The 15.8 mW power consumption is analyzed by Synopsys Design Compiler at 140 MHz operating frequency. Furthermore, the power consumption of each stage in the training and prediction

processes is analyzed using Synopsys PrimeTime in time-based mode. In terms of energy efficiency, the 2D node map with 8 actions per node achieves an energy consumption of 28 nJ, while the 3D node map with 10 actions per node consumes 35 nJ. For the abstract map, where each node is connected to 2–5 neighboring nodes, the energy efficiency is measured at 25 nJ/node.

##### B. Comparison with Prior Work

Compared to previous work, our processor achieves the highest success rate and energy efficiency. In a 2D 10×10 grid with 15 randomly placed obstacles, we conducted simulations over 100 trials on a set of 100 randomly generated obstacle maps with varying initial and goal locations, resulting in a total of 100,000 runs. As shown in Table I, our processor outperforms prior works with a higher success rate in 8 action directions. For energy efficiency in path planning, we tested the processor on a 2D 10×10 grid and a 3D 5×5×5 grid map with randomly placed obstacles. The results demonstrate a 1.2×–2.9× improvement in energy efficiency compared to prior processors, as shown in Table II.

#### V. CONCLUSION & FUTURE WORK

This paper presents a processor that leverages the advantages of CML to enable efficient self-supervised path planning, with a hardware architecture specifically designed to support on-chip training in dynamic map environments. Hardware-software co-design strategies are introduced: a CAM-based approach for associative memory access, which reduces memory accesses by up to 25%, and high-dimensional matrix sparsity optimizations, which reduce computational operations by over 13×. The processor is simulated in 40-nm CMOS technology, occupying an area of 2.88 mm<sup>2</sup> and achieving a power consumption of 15.8 mW at 140 MHz. Compared to prior works, the proposed processor demonstrates superior success rates and energy efficiency. Future work will focus on hardware designs that enable large-scale map navigation, enabling real-world applications.

#### ACKNOWLEDGEMENT

This work was supported in part by the National Natural Science Foundation of China under Grant 62476062; in part by the National Key Research and Development Program of China under Grant 2023YFB4704101, and in part by the Shanghai Platform for Neuromorphic and AI Chip (NeuHelium).

## REFERENCES

- [1] J. Lee, M. Bjelonic, A. Reske, L. Wellhausen, T. Miki, and M. Hutter, "Learning robust autonomous navigation and locomotion for wheeled-legged robots," *Science Robotics*, vol. 9, no. 89, p. eadi9641, 2024.
- [2] L. Yang, J. Jiang, X. Gao, Q. Wang, Q. Dou, and L. Zhang, "Autonomous environment-adaptive microrobot swarm navigation enabled by deep learning-based real-time distribution planning," *Nature Machine Intelligence*, vol. 4, no. 5, pp. 480–493, 2022.
- [3] C. Wang, X. Chen, C. Li, R. Song, Y. Li, and M. Q.-H. Meng, "Chase and track: Toward safe and smooth trajectory planning for robotic navigation in dynamic environments," *IEEE Transactions on Industrial Electronics*, vol. 70, no. 1, pp. 604–613, 2022.
- [4] D. Rachmawati and L. Gustin, "Analysis of dijkstra's algorithm and a\* algorithm in shortest path problem," in *Journal of Physics: Conference Series*, vol. 1566, no. 1. IOP Publishing, 2020, p. 012061.
- [5] A. Menon, A. Natarajan, L. I. G. Olascoaga, Y. Kim, B. Benedict, and J. M. Rabaey, "On the role of hyperdimensional computing for behavioral prioritization in reactive robot navigation tasks," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 7335–7341.
- [6] Z. Wan, A. Anwar, Y.-S. Hsiao, T. Jia, V. J. Reddi, and A. Raychowdhury, "Analyzing and improving fault tolerance of learning-based navigation systems," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 841–846.
- [7] Z. Wan, A. Anwar, A. Mahmoud, T. Jia, Y.-S. Hsiao, V. J. Reddi, and A. Raychowdhury, "Frl-fi: Transient fault analysis for federated reinforcement learning-based navigation systems," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 430–435.
- [8] H. Kwon, K. Kim, J. Lee, H. Lee, J. Kim, J. Kim, T. Kim, Y. Kim, Y. Ni, M. Imani *et al.*, "Brain-inspired hyperdimensional computing in the wild: Lightweight symbolic learning for sensorimotor controls of wheeled robots," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 5176–5182.
- [9] M. Nau, J. B. Julian, and C. F. Doeller, "How the brain's navigation system shapes our visual experience," *Trends in cognitive sciences*, vol. 22, no. 9, pp. 810–825, 2018.
- [10] J. C. Whittington, D. McCaffary, J. J. Bakermans, and T. E. Behrens, "How to build a cognitive map," *Nature neuroscience*, vol. 25, no. 10, pp. 1257–1272, 2022.
- [11] C. Stöckl, Y. Yang, and W. Maass, "Local prediction-learning in high-dimensional spaces enables neural networks to plan," *Nature Communications*, vol. 15, no. 1, p. 2344, 2024.
- [12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
- [13] Y. Kim, D. Shin, J. Lee, Y. Lee, and H.-J. Yoo, "A 0.55 v 1.1 mw artificial intelligence processor with on-chip pvt compensation for autonomous mobile robots," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 2, pp. 567–580, 2017.
- [14] C. Chung and C.-H. Yang, "A 1.5- $\mu$ j/task path-planning processor for 2-d/3-d autonomous navigation of microrobots," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 1, pp. 112–122, 2020.
- [15] S. Song, D. Han, S. Kim, S. Kim, G. Park, and H.-J. Yoo, "Gppu: A 330.4- $\mu$ j/task neural path planning processor with hybrid gnn acceleration for autonomous 3d navigation," in *2023 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*. IEEE, 2023, pp. 1–2.