Casper : Cascading Hypernetworks for Scalable Continual Learning

Tej Pandit
NUAI Lab
University of Texas at San Antonio
San Antonio, TX 78249
tej.pandit@utsa.edu

Dhireesha Kudithipudi
NUAI Lab
University of Texas at San Antonio
San Antonio, TX 78249
dk@utsa.edu

Abstract

Continual learning, the ability for a model to learn tasks sequentially without forgetting, remains a formidable challenge in deep learning. This paper introduces a novel approach, termed cascading hypernetworks, that combines the power of hypernetworks to generate the weights for multiple neural networks. To address the limited scalability of previous continual learning algorithms and accommodate an exponentially growing number of tasks, we propose a cascading architecture in which hypernetworks learn the weights of other hypernetworks. Additionally, with auto-generative replay, the hypernetwork generates samples of previous networks, mitigating forgetting without the need for an expanding memory buffer. Our findings highlight the promise of cascading hypernetworks in addressing the scalability and forgetting challenges inherent in continual learning, by evaluating their effectiveness on both reinforcement learning tasks and image classification benchmarks.

1 Introduction

While artificial intelligence (AI) has seen substantial progress, its ability to adapt to new tasks without compromising previously learned knowledge remains a critical challenge. This phenomenon, known as *catastrophic forgetting* [22], hinders the deployment of AI systems in real-world environments that demand continuous learning and adaptation [29].

1.1 Background

Several approaches have been proposed to address catastrophic forgetting [8, 18]. Regularization methods (e.g., EWC [16], SI [32], MAS [3]) restrict weight updates to preserve past knowledge. While they work effectively on small-scale networks, these methods struggle with scalability. Replay or rehearsal-based approaches [25, 28, 24], approximate interleaved learning by complementing the training data of the current task or experience with data that are representative of previous ones (e.g., GEM [21], A-GEM [14], DER[5]). They are promising but face challenges in memory management and potential bias [6, 19].

Hypernetworks [9] have emerged as a promising direction for continual learning (CL) due to their ability to generate weights for neural networks, offering a compact and flexible representation for knowledge transfer. Recent works such as continual learning with hypernetworks (HNet) [31] and partial hypernetworks (PHCL) [12] have shown the potential of hypernetworks for CL, in mitigating forgetting and in enabling efficient multi-task learning.

However, existing hypernetwork-based approaches [31, 12] still face limitations in terms of scalability and efficiency (as demonstrated in Fig. 2(a),(b)), particularly when dealing with a large number of tasks and complex task distributions. Moreover, reliance on an ever-expanding memory buffer (to accommodate new tasks) for replay can hinder their applicability in scenarios where memory resources are limited.

1.2 Contributions

We introduce a novel Cascading Hypernetwork architecture that enables scalable continual learning by leveraging the power of a series of hierarchical hypernetworks to generate the weights of the networks lower in the hierarchy. With **auto-generative replay** the hypernetwork regenerates previous network models, mitigating forgetting without the need for an expanding memory buffer and **node rearrangement** to reduce the interference among task-networks weights being learned. We demonstrate the effectiveness of the proposed technique on image classification and reinforcement learning tasks (RL), showcasing the ability to accommodate an exponentially growing number of tasks and networks.

2 Methodology

At the core of our approach is the hypernetwork. In addition to directly training the parameters (θ_T) in any target data distribution or task, also known as the task network model, we also train the parameters (θ_H) of a higher-order network also known as the hypernetwork. This in turn generates the parameters (θ_T) for the target model as required. Essentially, hypernetworks can be seen as weight generators. They were initially developed to enable dynamic and compressed parameterization of models, and we extend their use to accommodate the learning of multiple task oriented network models in a continual learning manner.

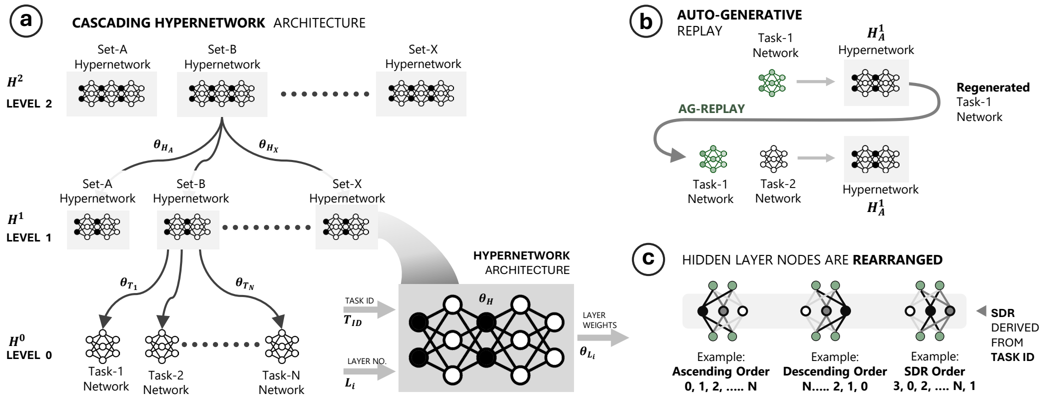

Figure 1: **(a) Cascading Hypernetwork Architecture**, **(b) Auto-Generative Replay**, **(c) Rearranging Network Nodes**

Instead of producing a high-dimensional output consisting of the entire set of weights, we instead rely on layerwise chunking [12] to iteratively generate the weights of an entire layer θ_{L_i} thus compressing the model size, at the cost of increased generation steps. Both the layer number L_i and the task ID T_{ID} are provided as context inputs to the hypernetwork as seen in Fig. 1(a).

As the hypernetwork continually learns task networks, it is itself subject to catastrophic forgetting. To mitigate this effect, we propose that the hypernetwork generate models of previously learned networks and replay these network samples alongside the current task network being learned (Fig. 1(b)). Given that the hypernetwork inherently functions as a generator, we eliminate the need for an external generator and instead leverage a mechanism we refer to as *auto-generative replay*.

Typically, generative replay methods require careful consideration of the generated samples' quality and diversity [17]. Our approach bypasses this concern, as the hypernetwork ideally aims to produce an exact replica of the original task networks. This simplifies the generative replay process and embraces overfitting, which is beneficial in this context.

In a continual learning setting, as the number of task networks being learned rises, the hypernetwork will invariably undergo a proportional degree of forgetting. Additionally, the buffer storing previous network samples will grow commensurately. To address this *hypernetwork saturation*, we propose demoting the hypernetwork itself to the role of a task network, with its weights learned by an even larger hypernetwork. This creates a *cascading hierarchy of hypernetworks*, where a new hypernetwork is introduced to learn a fixed number of tasks, and a parent hypernetwork is added to learn all such hypernetworks (Fig. 1(a)). This hierarchical structure assigns levels, starting with task networks at the base (level 0).

This approach serves to minimize the effects of catastrophic forgetting in hypernetworks by dividing the workload of learning multiple task networks into multi-level hierarchies which only grow upward as needed. Additionally, this serves to reduce the size of the replay buffer to the number of tasks per hypernetwork, rather than the total number of tasks.

Lastly, while Fig. 1(a) might suggest linear memory growth with more hypernetworks, our optimization strategy caps active networks at three. This includes the root hypernetwork (H^R), the current hypernetwork (H^N, optional), and the task network (H^0). In deeper hierarchies, H^R and H^N remain, while H^0 is replaced by the hypernetwork one level below (H^{N-1}). H^{N-1} then generates the next lower hypernetwork (H^{N-2}), replacing H^N in the process. This continues until the task network is reached.

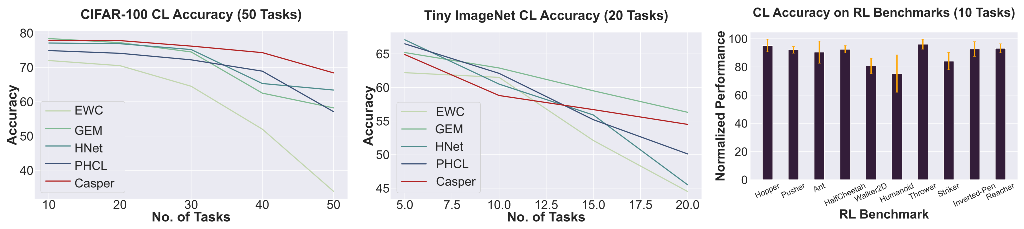

Figure 2: **Continual Learning Benchmarks (Task-IL) (a) CIFAR-100**, **(b) Tiny ImageNet**, **(c) Reinforcement Learning Mujoco Environments.**

While this process reduces the strain on individual hypernetworks themselves, it adds a reconstruction cost to the compute and reconstruction error to the generated network models (θ_w or θ_h). This error compounds at every level in the hierarchy, and is analyzed in Section 3. In order to mitigate the effects of reconstruction error, we try to identify the cause of this error. Reconstruction error grows based on the number of networks learned by a hypernetwork due to the differing distributions of weights in task networks. It has been observed that neural networks tend to find it more challenging to distinguish between similar tasks, as compared to distinct tasks [26]. As the hypernetwork learns the network models, the overlap between the weight distributions leads to confusion and thus increased forgetting. We propose a pre-processing step to reduce this overlap by rearranging the nodes of the network model.

As seen in Fig. 1(c), where the nodes of the hidden layer are rearranged, there is no change in the actual output of the network and the weights, inputs and edges remain connected in the same pattern. However, the numerical order of the nodes is shifted. This order does not affect network performance or operation. However, this order affects the learned representation within the hypernetwork. By creating maximal separation between the arrangement of subsequent task networks, the reconstruction error is minimized as seen in Section 3 (Fig. 3(b)).

The nodes are aligned based on their average cumulative incoming edge weights. This is trivial for small networks, but as the network dimension grows, calculating an optimum sorting order for N-networks incrementally becomes non-trivial. In this work, we propose the use of sparse distributed representations (SDR) [15] generated from task IDs to map the nodes of every task network in a non-overlapping manner. SDRs are inspired from the sparse neural processing that occurs in human brains [1], and have been formulated in spiking neural network architectures such as HTM [10, 7]. Alternatively non-spiking approaches such as generative models [13] and cellular automata [23] can also produce acceptable SDRs. In this paper we obtain a quantized linear mapping of the task ID as described in the Supplementary Materials.

3 Results

We evaluated our Cascading Hyper Networks (Casper) model on two image classification datasets, CIFAR-100 [2] and Tiny ImageNet [20], under a Task-IL continual learning scenario [30]. On CIFAR-100, partitioned into 50 tasks, each training a network on two classes, Casper demonstrated its ability to effectively learn numerous tasks with smaller network complexity. On Tiny ImageNet, with 20 tasks training networks on 10 classes each, Casper showcased its capability to handle larger network complexities in continual learning.

In comparison to state-of-the-art hypernetwork-based continual learning methods (HNet [31], PHCL [12]) and conventional approaches like replay-based methods [21] and EWC [16], Casper maintained comparable performance on both benchmarks (Fig. 2(a,b)). Notably, Casper outperformed these methods when the number of tasks increased substantially (≥ 20). While replay-based methods exhibited superior performance on Tiny ImageNet, this came at the cost of maintaining very large replay buffers.

The efficacy of Casper extended to reinforcement learning (RL) tasks in 3D physics environments (Mujoco-based Walker, Hopper, Ant) [27, 4]. When presented with 10 tasks sequentially in a continual learning fashion, Casper exhibited only a 4.57% loss in normalized accuracy compared to the original task performance prior to catastrophic forgetting. This highlights the potential of Casper in mitigating forgetting in complex continual learning RL environments (Fig. 2(c)). Varying hypernetwork model complexity on CIFAR-100 yielded a significant, linear performance gain across tasks. An average 3.6% improvement was observed per 10k additional parameters (Fig. 3(a)).

A key limitation identified in cascading hypernetworks is the compounding of errors over multiple levels. As a hypernetwork generates a task network, the final accuracy is influenced by both the original task network's accuracy and the reconstruction accuracy of the hypernetwork. This compounding can lead to accuracy degradation over levels, as illustrated in Fig. 3(c). To address this, we explored node rearrangement methods (Fig. 3(b)). While a random node structure resulted in poor reconstruction accuracy (89.1%), a complementary arrangement minimized accuracy degradation to $95.3 \pm 5.1\%$. However, computing unique complementary patterns for large networks is computationally intensive. Our proposed SDR-based rearrangement proved equally proficient but more reliable, demonstrating lower error margins $94.8 \pm 0.9\%$.

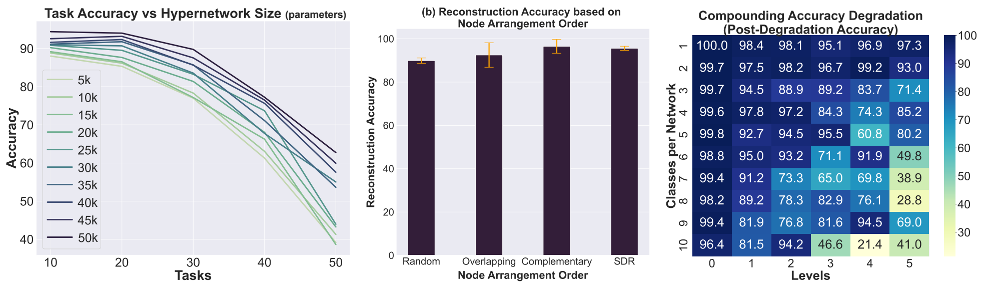

Figure 3: **(a) Impact of hypernetwork size on performance**, **(b) Reconstruction accuracy based on node arrangement order**, **(c) Compounding accuracy degradation across levels.**

4 Conclusion

This work introduces Cascading Hypernetworks, an innovative paradigm leveraging hypernetworks and auto-generative replay to tackle the challenge of continual learning. Our approach demonstrates superior performance and scalability in image classification tasks, as evidenced by average accuracy improvements on Cifar-100 $(+5.6\%)$ and Tiny Imagenet $(+3.4\%)$. These gains even extend to complex RL environments, where we saw an $+8.1\%$ improvement. This architecture mitigates catastrophic forgetting without the need for expansive memory, overcoming limitations of prior work. The cascading structure and auto-generative replay present a synergistic solution, enhancing the ability of models to continuously learn and adapt. Further exploration of this technique promises significant advancements in continual learning, facilitating the development of more scalable and intelligent AI systems.

Acknowledgments

This effort is partially supported by the NSF NAIAD award #2332744 and NSF EFRI BRAID award #2317706. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF, or the U.S. Government.

References

[1] Subutai Ahmad and Jeff Hawkins. How do neurons operate on sparse distributed representations? a mathematical theory of sparsity, neurons and active dendrites. *arXiv preprint arXiv:1601.00720*, 10, 2016.

[2] Krizhevsky Alex. Learning multiple layers of features from tiny images. *https://www. cs. toronto. edu/kriz/learning-features-2009-TR. pdf*, 2009.

[3] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European conference on computer vision (ECCV)*, pages 139–154, 2018.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[5] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. *Advances in neural information processing systems*, 33:15920–15930, 2020.

[6] Vivek Chavan, Paul Koch, Marian Schlüter, and Clemens Briese. Towards realistic evaluation of industrial continual learning scenarios with an emphasis on energy consumption and computational footprint. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11506–11518, 2023.

[7] Xi Chen, Wei Wang, and Wei Li. An overview of hierarchical temporal memory: A new neocortex algorithm. In *2012 Proceedings of International Conference on Modelling, Identification and Control*, pages 1004–1010, 2012.

[8] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3366–3385, 2021.

[9] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[10] Jeff Hawkins and Dileep George. Hierarchical temporal memory: Concepts, theory and terminology. *Numenta Inc*, 5, 2006.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] Hamed Hemati, Vincenzo Lomonaco, Davide Bacciu, and Damian Borth. Partial hypernetworks for continual learning. In *Conference on Lifelong Learning Agents*, pages 318–336. PMLR, 2023.

[13] Geoffrey E Hinton and Zoubin Ghahramani. Generative models for discovering sparse distributed representations. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 352(1358):1177–1190, 1997.

[14] Guannan Hu, Wu Zhang, Hu Ding, and Wenhao Zhu. Gradient episodic memory with a soft constraint for continual learning. *arXiv preprint arXiv:2011.07801*, 2020.

[15] Pentti Kanerva. *Sparse distributed memory*. MIT press, 1988.

[16] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[17] Alexander Krawczyk and Alexander Gepperth. An analysis of best-practice strategies for replay and rehearsal in continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4196–4204, 2024.

[18] Dhireesha Kudithipudi, Mario Aguilar-Simon, Jonathan Babb, Maxim Bazhenov, Douglas Blackiston, Josh Bongard, Andrew P Brna, Suraj Chakravarthi Raja, Nick Cheney, Jeff Clune, et al. Biological underpinnings for lifelong learning machines. *Nature Machine Intelligence*, 4(3):196–210, 2022.

[19] Dhireesha Kudithipudi, Anurag Daram, Abdullah M Zyarah, Fatima Tuz Zohora, James B Aimone, Angel Yanguas-Gil, Nicholas Soures, Emre Neftci, Matthew Mattina, Vincenzo Lomonaco, et al. Design principles for lifelong learning ai accelerators. *Nature Electronics*, 6(11):807–822, 2023.

[20] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

[21] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

[22] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[23] Tej Pandit and Dhireesha Kudithipudi. Low-shot learning and pattern separation using cellular automata integrated cnns. In *Proceedings of the International Conference on Neuromorphic Systems 2022*, pages 1–9, 2022.

[24] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[25] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[26] Rajat Saxena, Justin L Shobe, and Bruce L McNaughton. Learning in deep neural networks and brains with similarity-weighted interleaved learning. *Proceedings of the National Academy of Sciences*, 119(27):e2115229119, 2022.

[27] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[28] Gido M. van de Ven, Hava T. Siegelmann, and Andreas S. Tolias. Brain-inspired replay for continual learning with artificial neural networks. *Nature Communications*, 11(1):4069, Aug 2020.

[29] Gido M van de Ven, Nicholas Soures, and Dhireesha Kudithipudi. Continual learning and catastrophic forgetting. *arXiv preprint arXiv:2403.05175*, 2024.

[30] Gido M Van de Ven, Tinne Tuytelaars, and Andreas S Tolias. Three types of incremental learning. *Nature Machine Intelligence*, 4(12):1185–1197, 2022.

[31] Johannes Von Oswald, Christian Henning, Benjamin F Grewe, and João Sacramento. Continual learning with hypernetworks. *arXiv preprint arXiv:1906.00695*, 2019.

[32] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International conference on machine learning*, pages 3987–3995. PMLR, 2017.

A Appendix / Supplemental Material

A.1 Casper : Cascading Hypernetworks Algorithm

The pseudocode presented below (Algorithm 1) outlines the implementation of our proposed hypernetwork-based approach for continual learning. This method leverages the generative capabilities of hypernetworks to dynamically create task-specific models while mitigating catastrophic forgetting through auto-generative replay. The hypernetwork parameters (θ_H) are continually updated to generate parameters (θ_T) for both the current task network and previously learned task networks. The replay mechanism allows the hypernetwork to retain knowledge of past tasks, thereby enabling effective continual learning.

Algorithm 1 Casper

```
1: Set Root Level or No. of Levels, $N_l = 1$
2: Set Tasks-per-Network, $N_t$
3: Set Networks-per-Hypernetwork, $N_n$
4: Initialize Replay Memory Buffer, $M_r = [M_{r_0}, \dots M_{r_{N_n}}]$
5: Initialize Network Memory Buffer, $M_n = [M_{n_0}, M_{n_1}, M_{n_2}]$
6: Initialize Root Hypernetwork $H^2$ weights $\theta_H$
7: Store $M_{n_0} \leftarrow H^2$
8:
9: while tasks $t$ are available do
10:     for $n = 0, 1, 2, \dots, N_n$ do
11:         Initialize Task Network $H_n^0$ weights $\theta_T$
12:         for Every training epoch $\epsilon_t$ do
13:             Train $\theta_T$ using SGD or $\phi$-Policy (RL) in an environment for $T$ time steps
14:         end for
15:         Store $M_{r_n} \leftarrow H_n^0$
16:     end for
17:     Initialize Hypernetwork $H_x^1$ weights $\theta_H$
18:     for Every training epoch $\epsilon_h$ do
19:         Train $\theta_H$ using SGD on every network sample $H^0$ present in $M_r$
20:     end for
21:
22:     $L \leftarrow 1$
23:     while $L < N_l$ do
24:         $RecursiveNetworkGenerator(L, T_{ID})$
25:         Fill Replay Buffer with Hypernetworks $H_0^L, \dots, N_{max}$
26:         Store $[M_{r_0}, \dots, M_{r_{N_{max}}}] \leftarrow [H_0^L, \dots, H_{N_{max}}^L]$
27:         Restore $H^{L+1} \leftarrow M_{n_2}$
28:         Set Hypernetwork $H_{T[L]}^{L+1}$ weights as $\theta_H$
29:         for Every training epoch $\epsilon_h$ do
30:             Train $\theta_H$ using SGD on every network sample $H_x^L$ present in $M_r$
31:         end for
32:         Update $dict(heirarchy[L, T[L]]) \leftarrow N_{max}$
33:         Store $M_{r_{T[L]}} \leftarrow M_{n_2}$
34:     end while
35: end while
```

We separate the Recursive Network Generation process (Algorithm 2) from the main Casper algorithm (Algorithm 1) for increased clarity. Also, the Recursive Network Generator produces an end-point hypernetwork given the task-ID (T_{ID}) during the inference process (during which Casper is not required). After the algorithm runs, the final step during inference is the produce the required task network (H^0) from the final hypernetwork (H^1), and then use the task network to perform the required task.

The following table covers a breakdown of all the required terms mentioned in Algorithms 1 and 2. $RecursiveNetworkGenerator$ is the function call to Algorithm 2.

Algorithm 2 Recursive Network Generator

```
1: $L \leftarrow N_l$
2: $T \leftarrow SparseDistributedRepresentation(T_{ID})$
3: $T[1, 2...L] \leftarrow SDR\_partition(T, L)$
4: Restore $M_{n_0} \rightarrow H^L$
5: Copy $M_{n_1}, M_{n_2} \leftarrow M_{n_0}$
6:
7: while $L > 2$ do
8:     Move $M_{n_2} \rightarrow M_{n_1}$
9:     $N_{max} \leftarrow dict(heirarchy[L, T[L]])$
10:    Use $H^L$ in $M_{n_1}$ and $T[L]$ to generate $H_h^{L-1}$
11:    Store $M_{n_2} \leftarrow H_h^{L-1}$
12:    $L \leftarrow L - 1$
13: end while
14: Move $M_{n_2} \rightarrow M_{n_1}$
15:
16: for $n = 0, 1, 2, \ldots, N_max$ do
17:    Use $H^2_{T[L]}$ in $M_{n_1}$ and $T[L]$ to generate $H^1_n$
18:    Store $M_{r_n} \leftarrow H^1_n$
19: end for
```

A.2 Task-Network Details

For the task-network we use a standard ResNet-18 architecture, with pre-trained convolutional filters. However, this experimental methodology is extended to all other comparative models and benchmarks used in this test to avoid any bias.

A.3 Hypernetwork Details

In the construction of the hypernetwork, we employ a Multi-Layer Perceptron (MLP) architecture consisting of N-hidden layers. The task ID serves as the input to this hypernetwork. It is important to acknowledge that a naive approach to generating the weights of a model would involve generating all the weights, however this would require the hypernetwork to be of considerate size [12]. To avoid this, we only learn the parameters of the final fully connected (FC) layers of the model. The previous convolutional layers are pre-trained on the Imagenet Dataset. However, this experimental methodology is extended to all other comparative models and benchmarks used in this test to avoid any bias.

Every hidden layer has 100 hidden units and the number of layers (N) is determined by the layer number (L) in the hierarchy given by equation 1. B specifies the base model consisting of 2 hidden layers of 100 nodes each.

$$N = B + L \times 2 \tag{1}$$

A.4 Replay Memory Buffer Details

In this work we avoid storing data samples for replay (in raw or latent representations), we also do not generate the samples. Rather, we store the weights of learned networks in the replay memory buffer. However, this would also normally scale with the number of tasks being learned. To circumvent this, we have a fixed replay memory buffer of N_n memory slots M_r (equal to the number of networks per hypernetwork).

$$M_{RB} = (N_n + 3) \times mem(H^{N-1}) \tag{2}$$

$$AverageWeightedSum = \frac{\sum_{i=1}^{n} w_i \times i}{n} \tag{3}$$

Table 1: Algorithm Terminology

Term	**Definition**
N_l	Root Level or No. of Levels *This signifies the depth or number of layers in the hierarchical structure of hypernetworks.*
N_t	Tasks-per-Network *This represents the number of distinct tasks that a single task network (H^0) is expected to learn.*
N_n	Networks-per-Hypernetwork *This defines how many task networks (H^0) are generated or managed by a single hypernetwork (H^1 or higher).*
M_r	Replay Memory Buffer *This is a crucial storage mechanism that holds previously generated task networks (H^0) or hypernetworks (H^1 and above). It facilitates the training of higher-level hypernetworks by providing a fixed length (N_n) set of samples to learn from.*
M_n	Network Memory Buffer *This buffer specifically stores the current weights of a few key networks: the most recently generated task network or hypernetwork in M_{n_2}, the hypernetwork currently being trained in M_{n_1}, and the root or highest-level hypernetwork M_{n_0}.*
H^0	Task Network *This is the base-level network that directly interacts with the environment or data to learn and perform specific tasks.*
$H^1, \ldots, H^N$	Hypernetwork *This hypernetwork generates the weights or parameters for task networks (H^0).*
H^N	Root Hypernetwork *This is the top-level hypernetwork in the hierarchy, indirectly responsible for generating the weights of all the hypernetworks and task-networks in the hierarchy*
H^L	Hypernetwork at Level L
θ_H	Weights of a hypernetwork
θ_T	Weights of a task network
L	Current Level
T_{ID}	Task ID
SGD	Stochastic Gradient Descent
$\phi - Policy$	Reinforcement Learning Policy
ϵ_t	No. of training epochs for a task network
ϵ_h	No. of training epochs for a hypernetwork

In addition to this buffer there are 3 additional memory slots for storing networks M_n. Thus the size of the replay memory buffer M_{RB} can be given by equation 2. Additionally, the total memory M_{total} required by Casper is calculated by equation 4, by setting aside the root network H^N and the $BaseModel$ which is ResNet-18 [11] in this work.

As the size of the replay buffer is minimal and fixed to the N_n specification, we cannot perform an ablation study on the size of the buffer. However, we can analyze the effect of the degree of replay required while incrementally learning a new network. For both datasets, the accuracy is measured while varying both the percentage of network samples replayed while learning a new network and the number of networks assigned to each hypernetwork. The results observed in figure 4 show a steady performance being held for $> 60\%$ and $> 80\%$ replayed samples for CIFAR-100 and Tiny-ImageNet respectively. This is observed to be the case across denser network-to-hypernet configurations, albeit at slightly lower accuracies. After this point, there is a sharp decline in performance thereby indicating a significant amount of replay required for knowledge retention.

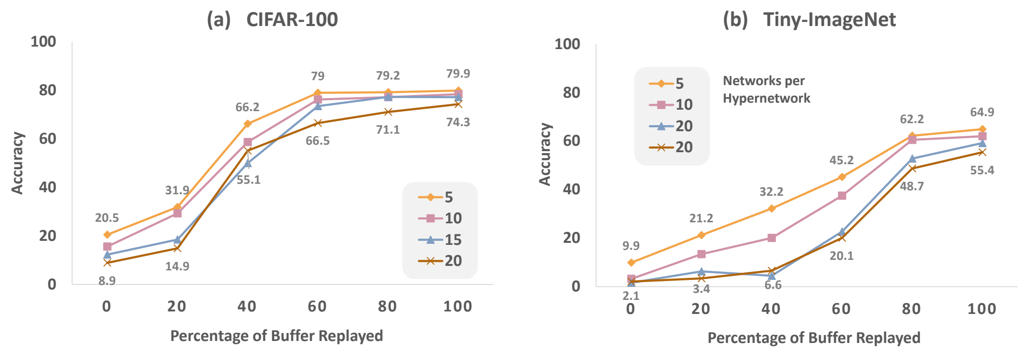

Figure 4: **Analysis of the effect of the percentage of the replay buffer network samples required to be replayed during Task-IL on (a) CIFAR-100 (b) Tiny-ImageNet, for architectures with a varying number of networks learned per hypernetwork.**

$$M_{total} = mem(H^N) + [(N_n + 2) \times mem(H^{N-1})] + mem(BaseModel) \tag{4}$$

A.5 Overlapping Nodes Algorithm

The average weighted sum at a node in a neural network, as calculated by equation 5, represents a unique aggregation of incoming signals. In addition to weighting each input by its connection strength, it also factors in the ordinal position or index (i) of the connection. This approach implies that the importance of an input increases linearly with its position in the sequence of connections. The equation calculates this weighted sum (AWS) by multiplying each weight value by its corresponding index, summing these products, and then dividing by the total number of connections

$$AverageWeightedSum = \frac{\sum_{i=1}^{n} w_i \times i}{n} \tag{5}$$

The AWS for every node in a layer is then re-ordered according to the representation produced by the SDR.

A.6 Sparse Distributed Representation Details

In order to transform a fixed-point binarized representation of a large integer (T_{ID}) into a sparse distributed representation, we begin by structuring the binary sequence into the rows of a square matrix. Subsequently, we employ a distance metric from the center of the matrix and the cosine of the angle with the horizontal axis for each “1” in the matrix to generate a new binary representation. This resultant binary representation constitutes the sparse distributed representation, or SDR. This approach leverages spatial properties within the matrix to encode the original integer into a high-dimensional, sparse format, offering advantages in terms of memory efficiency and pattern recognition capabilities.

However, this is a simplistic approach for SDR generation and unsuitable for RL tasks which require temporal information about the tasks. We prefer a relatively new approach of using cellular automata [23] to generate the SDR for temporal tasks. This work does not explicitly mention SDRs however the final state of the cellular automata after exposure to any spatio-temporal input results in a viable SDR. Especially since the training methodology prioritizes maximally separated output spaces, the resultant matrix after cellular automata computation will be sparse and unique to every temporal sequence i.e. task.

A.7 Compute Overheads : Retraining

After a new task-network is trained on a target task, the task-network can be learned by the parent hypernetwork. However, the replay buffer needs to be present during this training process. During online training, the previous immediate task-networks are already present in the replay buffer, and thus the generation compute cost is zero. However, when a random previous task (not a child of the current hypernetwork) recurs and needs to be retrained, the entire replay buffer of sibling task-networks need to be regenerated.

A.8 Compute Overheads : Task-Transition (Inference)

The greatest drawback of this method is the compute overhead and transition time delay while task switching during the inference phase. Unlike steady-state methods [16, 32] which do not require any change in the architecture while task-switching during inference, Casper requires the current task network to be regenerated. The Task-ID either explicitly provided or inferred by a SDR provides in the input for regeneration. Depending on the number of levels in the cascading hierarchy, the regeneration from root to task-network will require N_l full network regeneration cycles. This is a considerable overhead and can be time consuming. Thus, the Casper architecture is better suited to slow task-switching application domains.

A.9 Computing Resources and Experiments

All the training procedures carried out in the work were trained on a 12-core CPU workstation (i9-10920x) with 2 NVIDIA RTX 3090 GPUs serving as accelerators for the machine learning workload. The following average runtimes were recorded for the different benchmarks mentioned in the results section 3.

Table 2: Computing Metrics

Dataset	**Tasks/Network**	**Networks/Hypernet**	**Total Tasks**	**Training Time**
Cifar-100	2	5	50	346 mins
Tiny ImageNet	10	5	20	492 mins
Mujoco RL	1	2	10	644 mins