SHARED KNOWLEDGE LIFELONG LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

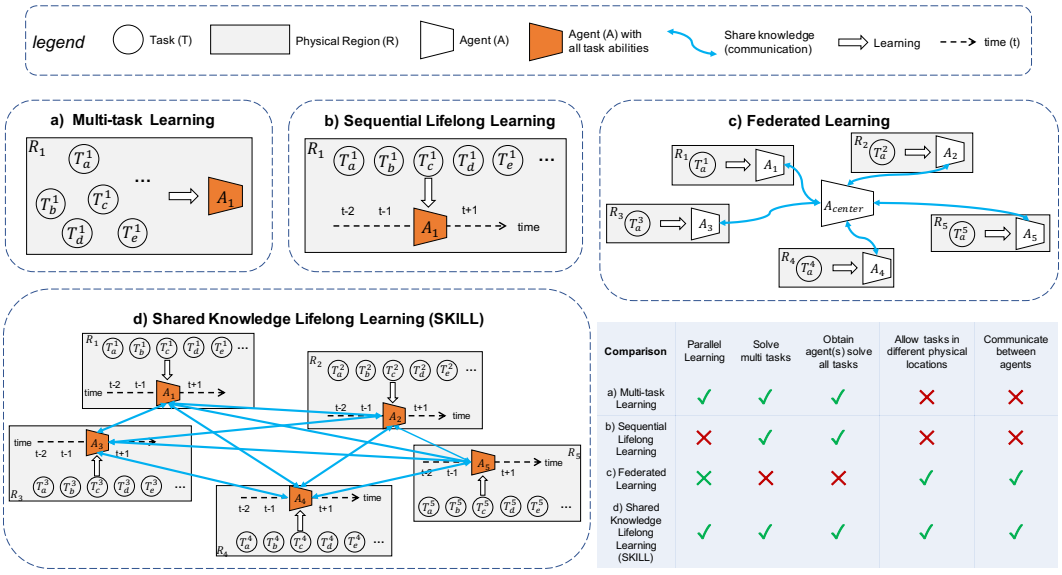
In Lifelong Learning (LL), agents continually learn as they encounter new conditions and tasks. Most current LL is limited to a single agent that learns tasks sequentially. Dedicated LL machinery is then deployed to mitigate the forgetting of old tasks as new tasks are learned. This is inherently slow. We propose a new Shared Knowledge Lifelong Learning (SKILL) learning paradigm, which deploys a population of LL agents that each learn different tasks independently and in parallel. After learning their respective tasks, agents share and consolidate their knowledge over a communication network, so that, in the end, all agents can master all tasks. Our approach relies on a frozen backbone embedded in all agents at manufacturing time, so that only the last layer *(head)* plus some small adjustments to the backbone *(beneficial biases)* are learned for each task. To eliminate the need for a task oracle, agents also learn and share summary statistics about their training datasets (Gaussian Mixture Clusters), or share a few training images, to help other agents assign test samples to the correct head using a Mahalanobis task mapper. On a new, very challenging dataset with 102 image classification tasks, we achieve significant speedup over 18 LL baselines (e.g., $> 9,000\times$ speedup over single-agent EWC) while also achieving higher (and SOTA) accuracy.

1 INTRODUCTION

Lifelong Learning (LL) is a relatively new area of machine learning (ML) research, in which agents continually learn as they encounter new tasks, acquiring novel task knowledge while avoiding forgetting of previous tasks (Parisi et al., 2019). This differs from standard train-then-deploy ML, which cannot incrementally learn without catastrophic interference across tasks French (1999).

Current LL research assumes a single agent that sequentially learns from its own actions and surroundings, which, by design, is not parallelizable over time and/or physical locations. In the real world, tasks may happen in different places; for instance, we may need agents that can run in deserts, forests and snow, as well as recognize birds in the sky and fish in the deep ocean. To solve the above challenges, we propose a new learning paradigm: Shared Knowledge Lifelong Learning (SKILL), which extends current LL to large numbers of originally identical agents. When agents are deployed, they may encounter different inputs and environmental conditions, execute different tasks, and therefore learn different knowledge. Other agents in the population could benefit if what is learned by one agent can be shared with other agents. Such sharing of knowledge could significantly reduce the amount of training required by any individual agent.

Our main contributions are: (1) A new learning paradigm (SKILL), which we contrast with multi-task learning, sequential LL, and federated learning (Sec. 2). (2) A new LL benchmark dataset: SKILL-102, with 102 complex image classification tasks. To the best of our knowledge, it is the most challenging benchmark to evaluate LL and SKILL algorithms in the classification domain, with the largest number of tasks, classes, and inter-task variance (Sec. 3). (3) A solution to the SKILL problem. For efficient knowledge sharing among agents, we use a fixed backbone plus task-specific head and beneficial biases, where knowledge is represented by small amounts of parameters and can be easily shared among agents, with negligible knowledge consolidation cost. The need for a task oracle is eliminated by using a task mapper, which can automatically determine the task (and corresponding head to use) at inference time, using Gaussian Mixture Clusters (GMMC) and Mahalanobis distance (Sec. 4). (4) Our SKILL algorithm achieves SOTA performance on three main metrics: High LL task accuracy (less catastrophic forgetting), low shared (communication) resources, and high speedup ratio, compared with 18 baselines (Sec. 5).



Comparison	Parallel Learning	Solve multi tasks	Obtain agent(s) solve all tasks	Allow tasks in different physical locations	Communicate between agents
a) Multi-task Learning	✓	✓	✓	✗	✗
b) Sequential Lifelong Learning	✗	✓	✓	✗	✗
c) Federated Learning	✗	✗	✗	✓	✓
d) Shared Knowledge Lifelong Learning (SKILL)	✓	✓	✓	✓	✓

Figure 1: SKILL vs. related learning paradigms. a) Multi-task learning (Caruana, 1997): one agent learns all tasks at the same time in the same physical location. b) Sequential Lifelong Learning (S-LL) (Li & Hoiem, 2017): one agent learns all tasks sequentially in one location, deploying LL-specific machinery to avoid task interference. c) Federated learning (McMahan et al., 2017): multiple agents learn *the same task* in different physical locations, then sharing learned knowledge (parameters) with a center agent. d) Our SKILL: different S-LL agents in different physical regions each learn tasks, and learned knowledge is shared among all agents, such that finally all agents can solve all tasks. Bottom-right table: Strengths & weaknesses of each approach.

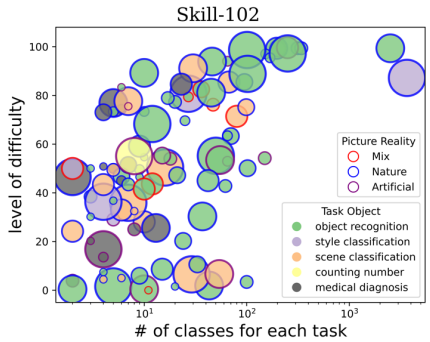
2 SHARED KNOWLEDGE LIFELONG LEARNING (SKILL)

Assumptions: (1) A population of N agents wants to learn a total of T different tasks separated into N physical regions. (2) Each agent i asynchronously learns $1 \leq T_i \leq T$ tasks from the distinct inputs and operating conditions it encounters. (3) Each agent performs as a *"teacher"* for its T_i tasks, by sharing what it has learned with the other $N-1$ agents; at the same time, each agent also performs as a *"student"* by receiving knowledge from the other $N-1$ agents. In the end, every agent has the knowledge to solve all T tasks. Fig. 1 contrasts SKILL with other learning paradigms.

Challenges: *What knowledge should be shared?* SKILL agents must share knowledge that is useful to other agents and avoid sharing local or specialized knowledge that may be misleading, in conflict with, or inappropriate to other agents. The shared knowledge may include model parameters, model structure, generalizations/specializations, input data, results of actions, specific contextual information, etc. There are also size/memory/communication constraints for the shared knowledge.

Evaluation metrics: **(1) Number N of agents and T of tasks.** For simplicity and without the loss of generality, in our experiments we focus on maximum parallelization, which is when $T = N$ and $\forall i, T_i = 1$. We consider $N = T = 102$ challenging image classification tasks, each consisting of learning to discriminate $2 \sim 300$ object classes. **(2) CPU/computation expenditure.** Wall-clock time is the main metric of interest, so that speedup can be achieved through parallelism. Thus, if N agents learn for 1 unit of time, wall-clock time would be 1, which is an N-fold speedup over a single sequential agent. In practice, speedup $< N$ is expected because of the overhead for sharing, communications, and knowledge consolidation. Because wall clock time assumes a given CPU or GPU speed, we instead report the number of multiply-accumulate (MAC) operations. **(3) Network/communication expenditure.** Sharing knowledge over a network is costly. To relate communications to computation, we assume a factor $\alpha = 1,000$ MACs / byte transmitted. **(4) Performance**, we use aggregated (averaged) performance over all T tasks (correct classification rate over all tasks). Note that there is *no task oracle at test time*. After training, agents should be able to handle any input from any task without being told which task that input corresponds to.

(a)



(b)

Comparison	SKILL-102	Visual domain decathlon	Cifar100	F-CelebA	Fine grained 6 tasks
# Tasks	**102**	10	20	10	6
# Classes	**10,418**	3,128	100	20	1,943
# Images	**3,325,244**	1,659,142	60,000	1,180	1,440,086
Max(Δ accuracy/difficulty)	**99.60%**	97.56%	27.45%	36.36%	49.50%
# Different classification target	**5**	2	1	1	2
Mix image style (nature/artifact)	✓	✓	✓	✗	✓
Mix super/fine-class classification	✓	✓	✗	✗	✓

Figure 2: (a) SKILL-102 dataset visualization. Task difficulty (y-axis) was estimated as the accuracy of a ResNet-18 trained from scratch on each task for a fixed number of epochs. Circle size reflects dataset size (number of images) (more dataset vis in Appen). (b) Comparison with other benchmark datasets including Visual Domain Decathlon (Ke et al., 2020), Cifar-100 (Krizhevsky et al., 2009), F-CelebA (Rebuffi et al., 2017a), Fine-grained 6 tasks (Russakovsky et al., 2014) (Wah et al., 2011), (Nilsback & Zisserman, 2008), (Krause et al., 2013), (Saleh & Elgammal, 2015), (Eitz et al., 2012)

3 SKILL-102 DATASET

We use image classification as the basic task framework and propose a novel LL benchmark dataset: SKILL-102 (Fig. 2). SKILL-102 consists of 102 image classification datasets. Each one supports one complex classification task, and the corresponding dataset was obtained from previously published sources (e.g., task 1: classify flowers into 102 classes, such as lily, rose, petunia, etc using 8,185 train/val/test images; task 2: classify 67 types of scenes, such as kitchen, bedroom, gas station, library, etc using 15,523 images; details in Appendix). Note that we have also successfully applied SKILL to visually-guided reinforcement learning (RL), using 54 Atari games (not shown here for brevity).

In total, SKILL-102 comprises $>$ 3.3M images in $>$ 10k classes over 102 tasks. For the experiments below, we subsampled the dataset slightly to allow some of the sequential baselines to converge: we capped the number of classes/task to 300 (only affected 4 tasks), and used either up to $5,120$ training images for tasks with $c \geq 60$ classes, or up to $2,560$ for tasks with $c < 60$. Thus, we used 102 tasks, total 4,785 classes, total 262,901 training images. After training, the algorithm is presented 27,757 test images and decides, for each image, which of the 4,785 classes it belongs to (no task oracle). To the best of our knowledge, SKILL-102 is the most challenging image classification benchmark for LL and SKILL algorithms, with the largest number of tasks, number of classes, and inter-task variance.

4 SKILL ALGORITHM DESIGN

Fig. 3 shows the overall pipeline and 4 roles for each agent. Agents share a common frozen backbone and only a "head" (last layer + beneficial biases) is trained per agent and then shared among agents. This makes the cost of both training and sharing very low. Receiving agents simply accumulate received heads and GMMC clusters in banks, and the GMMC clusters form a task mapper. At test time, we first run input data through the task mapper to recover the task, and then invoke the corresponding head to obtain the final system output. In addition to GMMC, we also implemented a Mahalanobis task mapper, which performs slightly better, at a higher sharing cost.

Pretrained backbone: We use the xception (Chollet, 2017) pretrained on ImageNet . The backbone is embedded in every agent at manufacturing time and is frozen. It processes 299×299 RGB input images, and outputs a 2048D feature vector.

Beneficial Biases: To address potentially large domain shifts between ImageNet and future tasks (e.g., line-drawing datasets, medical imaging datasets, space imaging datasets), we designed beneficial biases (BB). Inspired by BPN (Wen et al., 2021), BB provides a set of task-dependent, out-of-network bias units which are activated per task. These units take no input. Their constant outputs

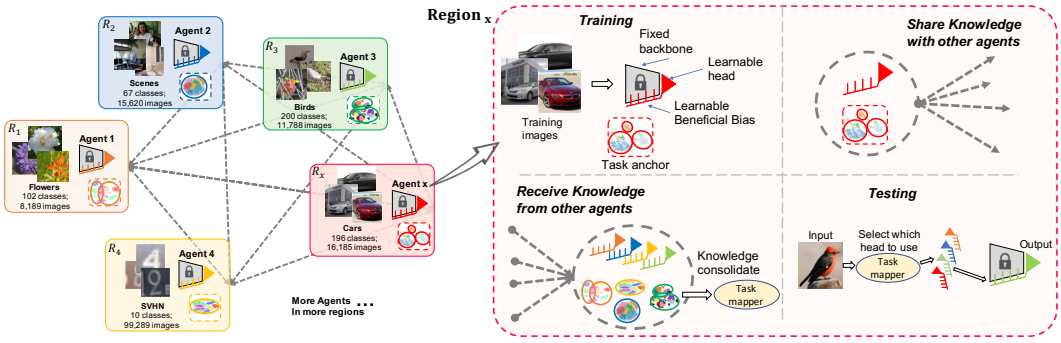


Figure 3: SKILL algorithm design. Left: overall pipeline, where agents are deployed in different regions to learn their own tasks. Subsequently, learned knowledge is shared among all agents. Right: Zoom into the details of each agent, with 4 main roles: **1) Training:** agents use a common pre-trained and frozen backbone, stored in ROM memory at manufacturing time (gray trapezoid with lock symbol). The backbone allows the agent to extract compact representations from inputs (e.g., with an xception backbone, the representation is a latent vector of 2048 dimensions, and inputs are 299×299 RGB images). Each agent learns a task-specific head (red triangle) for each new task. A head consists of the last fully-connected layer of the network plus our proposed LL beneficial biasing units that provide task-dependent tuning biases to all neurons in the network (one float number per neuron). During training, each agent also learns a GMMC or Mahalanobis task anchor which will form a task mapper. **2) Share knowledge** with other agents: except for the fixed backbone, each agent shares the learned task-specific head and GMMC module (or training images for Mahalanobis) with all other agents. **3) Receive knowledge** from other agents: each agent receives different heads and GMMC/Mahalanobis task mapper anchors from other agents. All heads are stored in a head bank and all task anchors are consolidated to form a task mapper. **4) Testing:** At test time, an input is first processed through the task mapper. This outputs a task ID, used to load up the corresponding head (last layer + beneficial biases) from the bank. The network is then equipped with the correct head and is run on the input to produce an output.

add to the biases of the neurons already present in the core network; thus, they provide one bias value per neuron in the core network. This is quite lightweight, as there are far fewer neurons than weights in the core network (22.9M parameters but only 22k neurons in xception). Different from BPN, which works best in conjunction with a constraint-type LL method like EWC Kirkpatrick et al. (2017) or PSP Cheung et al. (2019), and only works on fully-connected layers, BB does not require EWC or PSP, and can perform as an add-on module on both convolutional layers (Conv) and fully-connected layers (FC). Specifically, for each Conv layer, we have

$$y = Conv(x) + b + B \tag{1}$$

with input feature $x \in \mathbb{R}^{w*h*c}$, output feature $y \in \mathbb{R}^{w'*h'*c'}$. $b \in \mathbb{R}^{c'}$ is the original frozen bias of the backbone, and $B \in \mathbb{R}^{c'}$ is our learnable beneficial bias. The size of B is equal to the number of kernels (c') in this Conv layer. (w, h, c and w', h', c' denote the width, height and channels of the input and output feature maps respectively.) For FC layers,

$$y = FC(x) + b + B \tag{2}$$

with $x \in \mathbb{R}^{l}$, $y \in \mathbb{R}^{l'}$, $b \in \mathbb{R}^{l'}$ and $B \in \mathbb{R}^{l'}$. The size of B (beneficial bias) is equal to the number of hidden units (l') in this FC layer.

GMMC task mapper: To recover task at test time, each agent also learns Gaussian Mixture clusters (GMMC) that best encompass each of its task's data, and shares the clusters' parameters (means + diagonal covariances). This is also very fast to learn and very compact to share. As shown in Fig. 3(right), during training, each agent clusters its entire training set into k Gaussian clusters:

$$f(x) = \sum_{i=1}^{k} \phi_i \mathcal{N}(x|\mu_i, \Sigma_i), \qquad \sum_{i=1}^{k} \phi_i = 1 \tag{3}$$

We use $k = 25$ clusters for every task (ablation studies in Appendix). In sharing knowledge, each agent performs a "teacher" role on its learned task and shares the mean and diagonal covariance of

its clusters with all other agents (students). In receiving knowledge, each agent performs a "student" role and just aggregates all received clusters in a bank to form a task mapper with kT clusters, keeping track of which task any given cluster comes from: $D_{map}() = \{(\mathcal{N}_1, \phi_1) : 1, ..., (\mathcal{N}_{kT}, \phi_{kT}) : T\}$. At test time, a image x_i is evaluated against all clusters received so far, and the task associated with the cluster closest to the test image is chosen: $Task = D_{map}((\mathcal{N}_m, \phi_m))$, where $m = \arg\max_m(P(m, x_i))$. The probability of image x_i belonging to the m^{th} gaussian cluster is given by:

$$P(m, x_i) = \frac{\phi_m \mathcal{N}(x|\mu_m, \Sigma_m)}{\sum_{n=1}^{kT} \phi_n \mathcal{N}(x|\mu_n, \Sigma_n)} \tag{4}$$

Mahalanobis task mapper: To perform as a task mapper, the Mahalanobis distance (MD) method (Lee et al., 2018) learns C class-conditional Gaussian distribution $\mathcal{N}(x|\mu_c, \hat{\Sigma})$, c = 1,2, ... C, where C is the total number of classes of all T tasks and $\hat{\Sigma}$ is a tied covariance computed from samples from all classes. The class mean vectors and covariance matrix of MD are estimated as: $\mu_c = \frac{1}{N_c} \sum_{i:y_i=c} x_i$ (N_c: number of images in each class) and $\hat{\Sigma} = \frac{1}{N} \sum_{c=1}^{C} \sum_{i:y_i=c} (x_i - \mu_c)(x_i - \mu_c)^T$, ($N$: total number of images shared to the student agent). In training, each teacher agent computes the mean of each class within its task and randomly samples a variable number m of images per class. In our experiments, we use $m = 5$ for every task. During sharing knowledge, each agent shares the sample class means along with the saved images with all other agents. The shared images received by the student agents are used to compute the tied covariance. Similar to GMMC, the student agents also maintain a task mapper to keep track of which task any given class comes from. For a test image x, MD computes the Mahalanobis distance for all classes received so far and assigns the test image to the task associated with the smallest Mahalanobis distance, defined as:

$$\arg\min_c (x - \mu_c)^T \hat{\Sigma}^{-1} (x - \mu_c) \tag{5}$$

Implementation details: (1) Frozen xception backbone Chollet (2017), with 2048D latent representation. (2) Each agent learns one "head" per task, which consists of one fully-connected layer with 2048 inputs from the backbone and c outputs for a classification task with c classes (e.g., task 1 is to classify $c = 102$ types of flowers), and BB biases that allow us to fine-tune the backbone without changing its weights, to mitigate large domain shifts. (3) Each agent also fits $k = 25$ Gaussian clusters in the 2048D latent space to its training data. (4) At test time, a test image is presented and processed forward through the xception backbone. The GMMC classifier then determines the task from the nearest Gaussian cluster. The corresponding head is loaded and it produces the final classification result: which image class (among 4,785 total) the image belongs to. (5) The workflow is slightly different with the Mahalanobis task mapper: while GMMC clusters are learned separately at each teacher for each task as the task is learned, the Mahalanobis classifier is trained by students after sharing, using 5 images/class shared among agents. (6) Agents are implemented in pyTorch and run on desktop-grade GPUs (e.g., nVidia 3090, nVidia 1080). (More details in Appendix.)

5 Experiments and results

Baselines: Most baselines cannot be parallelized by design, so we use one agent to learn all SKILL-102 tasks in sequence. For those methods that require a task oracle, we (unfairly to us) grant them a perfect task oracle (while our approach uses imperfect GMMC or Mahalanobis). We implemented 18 baselines, which can be roughly categorized in the following 3 categories De Lange et al. (2021): *(1) Regularization methods* add an auxiliary loss term to the primary task objective to constraint weight updates. The extra loss can be a penalty on the parameters (EWC (Kirkpatrick et al., 2017), MAS (Aljundi et al., 2018) and SI (Zenke et al., 2017)) or on the feature-space (FDR (Benjamin et al., 2018)), such as using Knowledge Distillation (DMC (Zhang et al., 2020)). We use EWC as the representative of this category: one agent learns all 102 tasks in sequence, using EWC machinery to constrain the weights when a new task is learned, to attempt to not destroy performance on previously learned tasks. To give the best chances of success to this baseline, the whole xception is trained (not just the last layer). This takes a lot more training time, yet we will see below that performance is still below ours. *(2) Parameter-Isolation methods* assign a fixed set of model parameters to a task and avoid over-writing them when new tasks are learned (SUPSUP (Wortsman et al., 2020)), PSP (Cheung et al., 2019). We use PSP as the representative of this category: one agent learns all 102 tasks in sequence, generating a new PSP key for each task. The keys help segregate the tasks within the network in an attempt to minimize interference. Here again, the whole network

(backbone + head) is trained. We used the original PSP implementation, which uses a different backbone than ours. PSP accuracy overall hence is a bit lower because of this, and thus we focus on trends (decline in accuracy as more tasks are added) as opposed to only the absolute accuracy figures. *(3) Rehearsal methods* use a buffer containing sampled training data from previous tasks, as an auxiliary to a new task's training set. The buffer can be used either at the end of the task training (iCaRL, ER (Rebuffi et al., 2017b; Robins, 1995)) or during training (GSS, AGEM, AGEM-R, GSS, DER, DERPP (Lopez-Paz & Ranzato, 2017; Chaudhry et al., 2018; Aljundi et al., 2019; Buzzega et al., 2020)). We use DERPP and as the representative of this category: One agent learns all 102 tasks in sequence using a full (unfrozen) xception. After learning each task, it keeps a memory buffer with 10,000 images in total (GEM, A-GEM, A-GEM-R, DER, DERPP, ER, FDR, GEM, GSS, HAL) or 10 images/class of that task (Episodic Memory) that will later be used to rehearse old tasks. When learning a new task, the agent learns from all the data for that task, plus rehearses old tasks using the memory buffer.

Accuracy on first task: To gauge how well our approach is achieving LL (in a shared, parallelized manner), we plot the accuracy on the first task as we learn from 1 to 102 tasks, in Fig. 4. There is nothing special in our dataset about the first task, except that it is the first one. A good LL system is expected to maintain its accuracy on task 1 even as more subsequent tasks are learned; conversely, catastrophic interference across tasks would rapidly decrease task 1 accuracy with more learned tasks. Overall, our approach maintains the highest accuracy on task 1 over time, and virtually all of the accuracy degradation over time is due to increasing confusion in the task mapper (e.g., curves for Mahalanobis task mapper alone and SKILL w/BB w/MD are nearly shifted versions of each other). Indeed, once the task is guessed correctly, the corresponding head always performs exactly the same, no matter how many tasks have been learned.

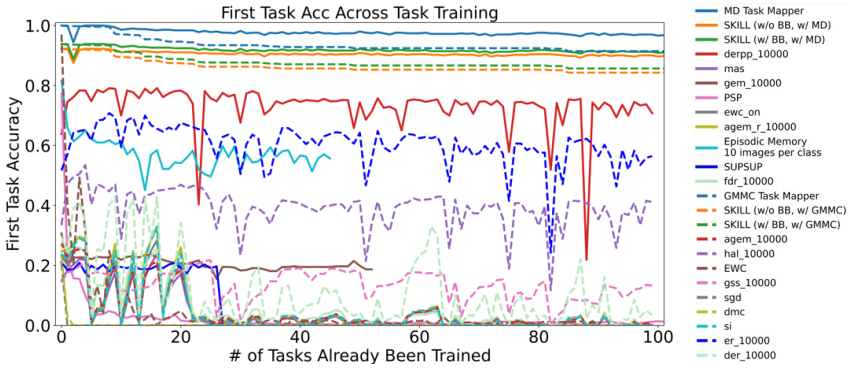


Figure 4: Accuracy on task 1 as a function of the number of tasks learned. Our approach is able to maintain accuracy on task 1 much better than the baselines as more and more tasks are learned: while our approach does suffer some interference, task 1 accuracy remains to within 90% of its initial best even after learning 101 new tasks (for the 4 SKILL variants, BB=beneficial biases, MD=Mahalanobis Distance task mapper, GMMC=GMMC task mapper). In contrast, the accuracy of EWC, PSP, and several other baselines on task 1 catastrophically degrades to nearly zero after learning just 20 new tasks. Best performing baselines are of the episodic buffer type (a fraction of the training set of each task is retained for later rehearsing while learning new tasks), with a large buffer of 10,000 images. These methods do incur higher (and increasing) training costs because of the rehearsing, as studied in the next section.

Normalized accuracy on first 10 tasks: We compare our method to the baselines on the first 10 tasks, when up to 20 subsequent tasks are learned. A good LL system should be able to maintain accuracy on the first 10 tasks, while at the same time learning new tasks. Because in SKILL-102 different tasks have different levels of difficulty, we normalize accuracy here to focus on degradation with an increasing number of new tasks. For example, the accuracy of our method (SKILL w/o BB) when learning a single task is 89.80% for task 1, but only 40.82% for task 4, which is much harder. Here, we define a **normalized accuracy** as the accuracy divided by the initial accuracy just after a given task was learned (which is also the best ever accuracy obtained for that task). This way, normalized accuracy starts at 100% for all tasks. If it remains near 100% as subsequent tasks are learned, then the approach is doing a good job at minimizing interference across tasks. Conversely,

a rapidly dropping normalized accuracy with an increasing number of subsequent tasks learned indicates that catastrophic interference is happening.

Our results in Fig. 5 show that, although not perfect, our approach largely surpasses the EWC, PSP, and Episodic Buffer baselines in its ability to maintain the accuracy of previously learned tasks.

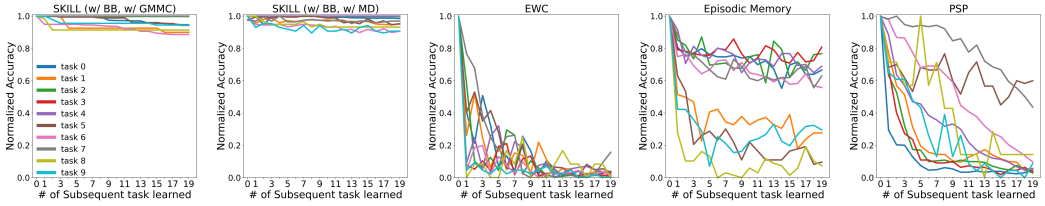


Figure 5: Normalized accuracy on the first 10 tasks (one per curve color) as up to 20 additional tasks are learned. Our SKILL approach (left two charts) is able to maintain high normalized accuracy on the first 10 tasks, while the EWC, PSP and Episodic Buffer baselines (right 3 panels) suffer much stronger catastrophic interference.

Average normalized accuracy on all tasks after learning 1 to 102 tasks: We computed the average normalized accuracy on all tasks learned so far, after learning 1, 2, 3, ... 102 tasks, in Fig. 6: It starts at 100% after learning 1 task (all test samples are correctly assigned to that task by GMMC or Mahalanobis, and normalized accuracy of xception+head is 100% when there is only one task), then decreases as more tasks are learned, eventually still achieving 81.42% correct after 102 tasks.

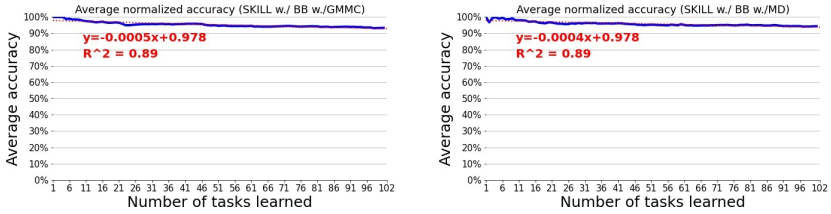


Figure 6: Average normalized whole-system accuracy on all tasks learned so far, as a function of the number of tasks learned, when using GMMC (left) or Mahalanobis (right) task mappers. Our approach is able to maintain average accuracy as the number of tasks increases.

When using GMMC task mapping, the regression line is $y = -0.0005x + 0.978$, which intercepts zero for $N = 1956$. Thus, with the difficulty level of our dataset, we extrapolate that $N = 1500$ is realistic as is. Since task interference in our system comes from GMMC, pushing beyond $N = 1500$ might require more than $k = 25$ GMMC clusters per task, which would increase CPU expenditure a bit. When using Mahalanobis task mapping, the results are similar with an intercept at $N = 2445$, though this approach incurs a higher communications cost.

Absolute accuracy: The normalized accuracy figures reported so far were designed to factor out variations in individual task difficulty, so as to focus on degradation due to interference among tasks. However, they also factor out the potential benefits of BB in raising absolute task accuracy. Hence, we here also study absolute task accuracy.

We first plot the absolute accuracy for the GMMC component and for the xception+head component (without or with BB) in Fig. 7. This shows that our SKILL-102 dataset provides a range of difficulty levels for the various tasks, and is quite hard overall.

We then plot the absolute accuracy averaged over all tasks learned so far in Fig. 8. The absolute accuracy for GMMC and Mahalanobis is the same as before. However, now the absolute accuracies for the full SKILL models and for the baselines conflate two components: 1) how much interference exists among tasks and 2) the absolute difficulty of the tasks learned so far.

Computation and communication costs, SKILL metrics: The baselines are sequential in nature, so trying to implement them using multiple agents does not make sense as it would only add communication costs but not alleviate the sequential nature of these LL approaches. For example, for

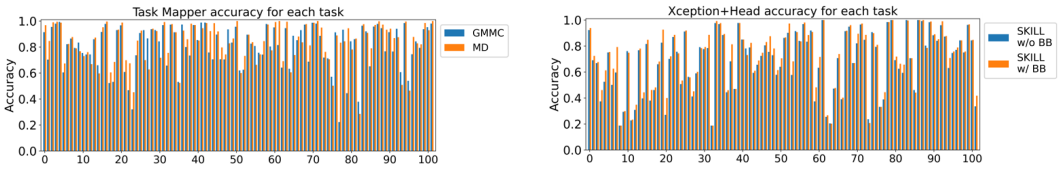


Figure 7: Absolute accuracy per task after learning 102 tasks. (left) Absolute accuracy of the GMMC and Mahalanobis task mappers alone shows quite a bit of variability, indicating various degrees of overlap among tasks. (right) Absolute accuracy of the main xception+head network alone (with or without BB, assuming perfect GMMC) also shows significant variability, indicating various degrees of difficulty per task. Although not obvious here, the accuracy with BB is overall slightly higher than without BB, as further explored in the next figure.

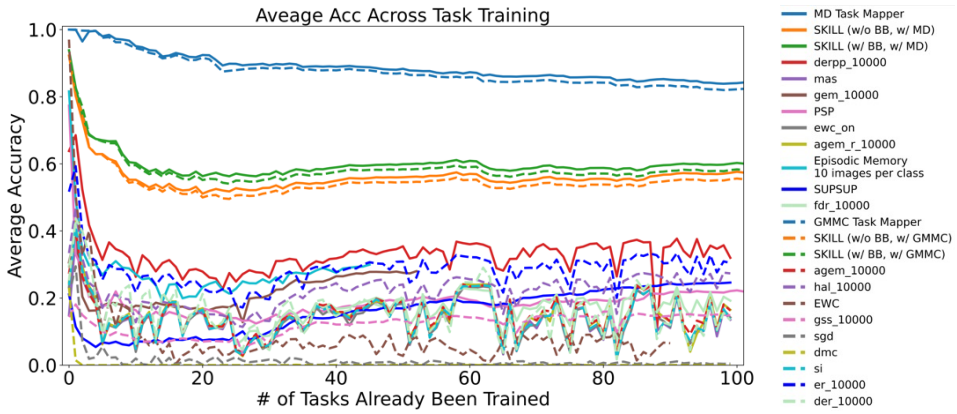


Figure 8: Average absolute accuracy on all tasks learned so far, as a function of the number of tasks learned. Our SKILL approach is able to maintain higher average accuracy than all baselines. BB provides a small but reliable performance boost (SKILL w/BB vs. SKILL w/o BB). The sharp decrease in early tasks carries no special meaning except for the fact that tasks 3-5 are significantly harder than tasks 1-2, given the particular numbering of tasks in SKILL-102.

the EWC baseline, one could learn task 1 on agent A then communicate the whole xception weights to agent B (22.9 M parameters = 91.6 MBytes) plus the diagonal of the Fisher Information matrix (another 22.9 M parameters), then agent B would learn task 2 and communicate its resulting weights and Fisher matrix to agent C, etc. Agent B cannot start learning task 2 before it has received the trained weights and Fisher matrix from agent A because EWC does not provide a mechanism to consolidate across agents. Thus, we set the communications cost to zero for the baselines and consider that each baseline runs on a single agent, learning all 102 tasks sequentially.

In our approach, we use $N = 102$ agents that each learn 1 task. All agents learn in parallel. Each agent is the "teacher" for its assigned task, and "student" for the other $N - 1$ tasks. Then all agents broadcast their shared knowledge to all other agents. As they receive shared knowledge, the students just accumulate it in banks, and possibly update their task mapper. After sharing, all agents know all tasks (and are all identical). As mentioned above, the main source of performance degradation in our approach is in the task mapper, which gets increasingly confused at N increases.

Fig. 9 shows the computation and network expenditures for our approach and the baselines to learn SKILL-102 dataset. Because some algorithms run on GPU (e.g., xception backbone) but others on CPU (e.g., GMMC training), and because our tasks use datasets of different sizes, we measure everything in terms of MACs (multiply-accumulate operations, which are implemented as one atomic instruction on most hardware). To translate communication costs to MACs, we assume a nominal cost of $\alpha = 1,000$ MACs to transmit one byte of data.

Our results in Fig. 9 show: **(1) Our approach has very low parallelization overhead**, which leads to almost perfect speedup $> 0.99N$ for GMMC (with or without BB), and good speedup $> 0.79N$ for Mahalanobis (which has more communication and student costs, but also slightly better accu-

102 tasks, single agent	Teacher CPU (MACs)	Communications (MACs equivalent)	Student CPU (MACs)	Total (MACs)	Speedup (factor xN)	Avg Accuracy (102 test sets)	Slow-down factor vs. ours-SKILL, w/o BB, w/ GMMC
Ours-single, w/o BB, w/ GMMC	2.2232E+15	0	0	2.2232E+15		**55.40%**	
Ours-single, w/ BB, w/ GMMC	2.5075E+15	0	0	2.5075E+15		**58.17%**	
Ours-single, w/o BB, w/ MD	2.2220E+15	0	5.9813E+09	2.2220E+15		**57.38%**	
Ours-single, w/ BB, w/ MD	2.5050E+15	0	5.9813E+09	2.5050E+15		**60.07%**	
EWC	2.0617E+17	0	0	2.0617E+17		**6.53%** (91/102 completed)	9424.0
PSP	8.0820E+16	0	0	8.0820E+16		**21.95%**	3694.3
Episodic Memory	5.8970E+17	0	0	5.8970E+17		**29.95%** (46/102 completed)	26955.4
DERPP	9.7254E+17	0	0	9.7254E+17		**32.00%** (100/102 completed)	44455.4
102 tasks, 102 SKILL agents							
Ours-SKILL, w/o BB, w/ GMMC	2.1797E+13	8.0203E+10	0	2.1877E+13	0.9963	**55.40%**	
Ours-SKILL, w/ BB, w/ GMMC	2.4583E+13	1.0053E+11	0	2.4684E+13	0.9959	**58.17%**	
Ours-SKILL, w/o BB, w/ MD	2.1784E+13	6.3927E+12	5.9813E+09	2.8183E+13	0.7730	**57.38%**	
Ours-SKILL, w/ BB, w/ MD	2.4558E+13	6.4130E+12	5.9813E+09	3.0977E+13	0.7928	**60.07%**	

Figure 9: Analysis of computation and network expenditures for our approach and 4 representative baselines, to learn all 102 tasks. **Teacher CPU:** Our single agents learn 102 tasks in sequence, while each of our 102 SKILL agents learns only 1 task; hence teacher CPU time for single agents is $102\times$ that for SKILL agents. BB adds to training time, to compute the biases. Our teacher CPU is much lower than most baselines because we only train the last layer, while most baselines train all layers: this is $\sim 89.7\times$ slower for EWC and Episodic Buffer, and $36.4\times$ slower for PSP, which uses a smaller network. The baselines then also incur extra LL costs (computing the Fisher matrix for EWC, PSP keys for PSP, and rehearsing old tasks for Episodic Buffer). **Communications:** Single agents do not have communication costs. SKILL agents communicate either GMMC clusters or Mahalanobis training images. Here we assume that there is a communication bottleneck at the receiver (student): the shared data from 101 tasks needs to be received serially, over a single networking interface for each student. Hence our communication figures are for all the shared data from all other 101 tasks apart from the one a SKILL agent learned itself. We convert communication costs to equivalent MACs by assuming 1,000 MACs per byte transmitted. BB adds a small extra communication cost, to transmit the biases. **Student CPU:** For GMMC, students do not do any extra work (hence, student CPU is 0); for Mahalanobis, students compute a covariance matrix for all 102 tasks. **Speedup factor:** is just total MACs for single agent divided by total MACs for SKILL agent and by N. **Accuracy:** In addition to being faster, our SKILL approach also outperforms baselines, as detailed in several figures above. Accuracy for baselines is from Fig. 8. **Slowdown factor for baselines:** this is just total MACs for each baseline divided by total MACs for our SKILL method with GMMC, no BB. **Why are baselines so slow?** First they train all layers, which is necessary for these methods and/or is aimed to improve their accuracy, but which is also $89.7\times$ slower for EWC or Episodic Buffer, and $36.4\times$ slower for PSP, than training the last layer (head) only in our agents. Then they use 1 GPU while we parallelize over 102. So that is a factor $\sim 9,000$ already for EWC and Episodic Buffer, or $\sim 3,600$ for PSP. The rest is higher LL overhead compared to our approach (EWC Fisher matrix, PSP keys, Episodic rehearsing of old tasks). Additional details in Appendix.

racy). Indeed, teachers just learn their task normally, plus a small overhead to train GMMC on one task per teacher, when GMMC is used. Communications are just a few hundred KBytes (GMMC) or a few MBytes (Mahalanobis) per task (Appendix). Students either do nothing (just accumulate received knowledge in a bank) or update their Mahalanobis task mapper. **(2) The baselines have comparatively much higher training cost, yet their performance is poor.** Performance of episodic buffer / rehearsing methods can be improved further by increasing buffer size, but note that in the limit (keeping all training data for future rehearsing), this gives rise to a $5,000\times$ increase in training time (Appendix). **(3) One may argue that parallelism should be taken out of the slowdown factor for the baselines.** Then, instead of reporting raw slowdown factors (how much slower a baseline is compared to our SKILL approach), one could also factor out N. In this case, EWC is $92.4N$ slower than SKILL solution, PSP $36.2N$ slower, and Episodic Buffer $264.4N$ slower. But the flipside of this argument is: if 102 GPUs were available, who would use them? All baselines would only be able to use one, while SKILL takes advantage of all 102 with near-perfect speedup.

6 Conclusions

We have proposed a new framework for shared-knowledge, parallelized LL. The approach works much better than previously SOTA baselines, and is much faster. Scaling to $N > 1500$ difficult tasks like the ones in our new SKILL-102 dataset seems achievable with the current implementation.

REFERENCES

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)*, pp. 139–154, 2018.

Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. *Advances in neural information processing systems*, 32, 2019.

Ari S Benjamin, David Rolnick, and Konrad Kording. Measuring and regularizing networks in function space. *arXiv preprint arXiv:1805.08289*, 2018.

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.

Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018.

Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International conference on machine learning*, pp. 2285–2294. PMLR, 2015.

Brian Cheung, Alex Terekhov, Yubei Chen, Pulkit Agrawal, and Bruno Olshausen. Superposition of many models into one. *arXiv preprint arXiv:1902.05522*, 2019.

François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1251–1258, 2017.

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.

Mathias Eitz, James Hays, and Marc Alexa. How do humans sketch objects? *ACM Transactions on graphics (TOG)*, 31(4):1–10, 2012.

Gamaleldin F Elsayed, Ian Goodfellow, and Jascha Sohl-Dickstein. Adversarial reprogramming of neural networks. *arXiv preprint arXiv:1806.11146*, 2018.

Utku Evci, Vincent Dumoulin, Hugo Larochelle, and Michael C Mozer. Head2toe: Utilizing intermediate representations for better transfer learning. *arXiv preprint arXiv:2201.03529*, 2022.

Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

Tommaso Furlanello, Zachary Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born again neural networks. In *International Conference on Machine Learning*, pp. 1607–1616. PMLR, 2018.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Zixuan Ke, Bing Liu, and Xingchang Huang. Continual learning of a mixed sequence of similar and dissimilar tasks. *Advances in Neural Information Processing Systems*, 33:18493–18504, 2020.

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.

Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE international conference on computer vision workshops*, pp. 554–561, 2013.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, 2009.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990.

Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. *Advances in neural information processing systems*, 31, 2018.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Yuhang Li, Xin Dong, and Wei Wang. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. *arXiv preprint arXiv:1909.13144*, 2019.

Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

Peter Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE transactions on visualization and computer graphics*, 20(12):2674–2683, 2014.

Hugo Liu and Push Singh. Conceptnet—a practical commonsense reasoning tool-kit. *BT technology journal*, 22(4):211–226, 2004.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30:6467–6476, 2017.

Eran Malach, Gilad Yehudai, Shai Shalev-Schwartz, and Ohad Shamir. Proving the lottery ticket hypothesis: Pruning is all you need. In *International Conference on Machine Learning*, pp. 6682–6691. PMLR, 2020.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pp. 1273–1282. PMLR, 2017.

George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11): 39–41, 1995.

Ashvin Nair, Murtaza Dalal, Abhishek Gupta, and Sergey Levine. Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pp. 722–729. IEEE, 2008.

German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

Simone Parisi, Aravind Rajeswaran, Senthil Purushwalkam, and Abhinav Gupta. The unsurprising effectiveness of pre-trained vision models for control. *arXiv preprint arXiv:2203.03580*, 2022.

Rémy Portelas, Cédric Colas, Lilian Weng, Katja Hofmann, and Pierre-Yves Oudeyer. Automatic curriculum learning for deep rl: A short survey. *arXiv preprint arXiv:2003.04664*, 2020.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. volume 30, 2017a.

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*, pp. 2001–2010, 2017b.

Amanda Rios and Laurent Itti. Closed-loop memory gan for continual learning. *arXiv preprint arXiv:1811.01146*, 2018.

Anthony Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2): 123–146, 1995.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge (2014). *arXiv preprint arXiv:1409.0575*, 2(3), 2014.

Babak Saleh and Ahmed Elgammal. Large-scale classification of fine-art paintings: Learning the right metric on the right feature. *arXiv preprint arXiv:1505.00855*, 2015.

Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. Dataset distillation. *arXiv preprint arXiv:1811.10959*, 2018.

Shixian Wen, Amanda Rios, Yunhao Ge, and Laurent Itti. Beneficial perturbation network for designing general adaptive artificial intelligence systems. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. Supermasks in superposition. *Advances in Neural Information Processing Systems*, 33:15173–15184, 2020.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pp. 3987–3995. PMLR, 2017.

Junting Zhang, Jie Zhang, Shalini Ghosh, Dawei Li, Serafettin Tasci, Larry Heck, Heming Zhang, and C-C Jay Kuo. Class-incremental learning via deep model consolidation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 1131–1140, 2020.

A APPENDIX

A.1 SKILL-102 DATASET VS. OTHER BENCHMARKS

Fig. 10 compared the range of tasks in SKILL-102 to that of other popular benchmark dataset collections.

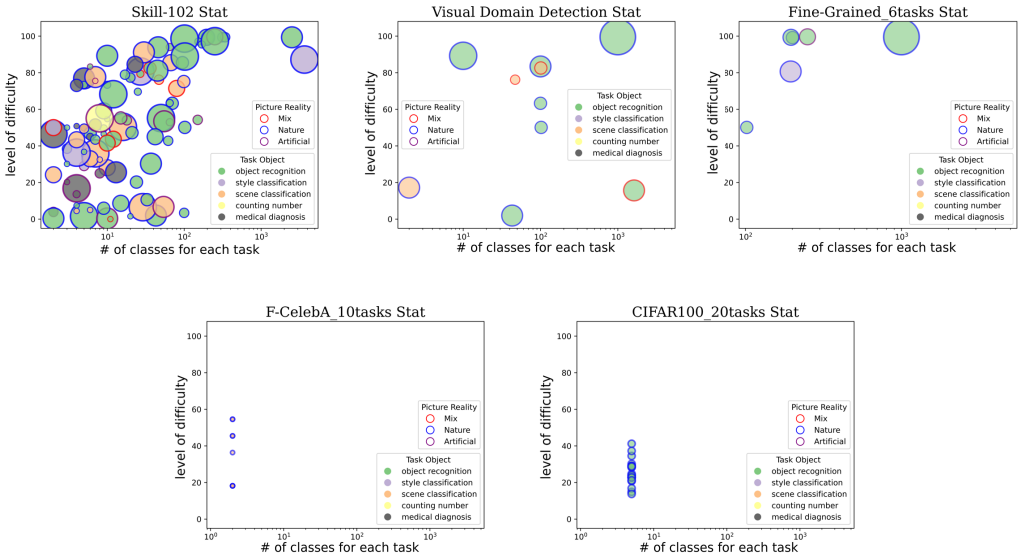


Figure 10: Visualization of SKILL-102 and other benchmark datasets.

A.2 ALTERNATIVES

Many other approaches may seem, at first, suitable to solve SKILL. Here we summarize a few. Note that in this project we are only interested in the approaches that would yield significant parallelization speedup. Hence we set the bar at:

Speedup with N agents must be $\geq 0.5N$.

We present estimates of computation and communication for alternative SKILL approaches, under the following core assumptions:

- Consider a dataset $\mathcal{D}$ to be learned by a collection of N SKILL agents. We first split this dataset (for now, equally) into N subsets $\mathcal{D}_1, ...\mathcal{D}_N$, each of size $|\mathcal{D}_i| = |\mathcal{D}|/N = D$ bytes.
- Each agent i assumes the role of Teacher for $\mathcal{D}_i$ (teacher learns from $\mathcal{D}_i$ and then shares with all other agents $j \neq i$). It also assumes the role of Student for $\mathcal{D}_{j,j\neq i}$ (students receive shared knowledge).
- Agent i, as a teacher, (L)earns from $\mathcal{D}_i$ in a CPU time L_t (assumed same for all agents, since all $\mathcal{D}_i$ have same size), then possibly does extra (P)rocessing to enable sharing, e.g., distills $\mathcal{D}_i$ into a small set of super-exemplars to share with other agents, in CPU time P_t (also assumed same for all agents). All agents learn and process their assigned D_i in parallel. Below we report $(L_t + P_t)/L_t$ as total time for the Teacher role in a normalized time unit that factors out dataset size, model architecture, and CPU speed. The resulting amount of data to be shared has size Z bytes (again assumed to be same for all agents; e.g., 10 distilled images). We report Z as a fraction of D; thus if agent i shares its whole $\mathcal{D}_i$, then $Z = D$ and $Z/D = 1$.
- Every agent i broadcasts its processed data of size Z to all other agents $j \neq i$ in parallel (assuming homogenous network links for now). We do not yet fix a specific relative time scale between computation and communications and instead use a factor α such that transmitting data of size D (whole dataset $\mathcal{D}_i$) takes as much time as α times the time required for learning from that dataset in the teacher, L_t.

- Every agent i, now as a Student, receives data of size Z from each of the $N-1$ other agents $j \neq i$, and learns from it in a CPU time L_s per dataset, which we again express relative to L_t. If Z/D is small, then we expect $L_s/L_t \ll 1$, i.e., the student learns much faster than the teacher. To learn from all other agents hence requires $(N-1)L_s/L_t$ normalized time in each student. We assume that all students learn in parallel.

Thus, total SKILL time is $(L_t+P_t)/L_t$ (normalized CPU time for teacher role, all agents in parallel) + $\alpha Z/D$ (communications) + $(N-1) \times L_s/L_t$ (students learn from $N-1$ teachers). Since each teacher only learns $1/N$ of $\mathcal{D}$, speedup hence is $1/[(L_t + P_t)/(N \times L_t) + \alpha Z/D + (N-1) \times L_s/(N \times L_t)]$, which we want to be $\geq 0.5N$.

Below we use $N = 100$ (number of agents, **so, 0.5N is 50x**), $\alpha = 0.01$ (transmitting data is 100x faster than learning from it), $\epsilon = 0.001$ (negligible operations add 0.1% to learning time). **An interactive spreadsheet will be provided at press time to allow anyone to change these values.** A good rule of thumb is that if a method more than doubles total teacher + student CPU, then no matter how small the transmitted data, that method alone will not achieve $0.5N$ speedup. Yet, methods below which alone do not achieve $0.5N$ but achieve a speedup > 1 can be combined with others as they overall contribute somewhat to speedup; e.g., dataset distillation is very slow in its vanilla implementation, but if we first apply OOD then distill, both together may look promising.

In this project, we had set out to explore alternatives as sub-tasks given to our students, as follows:

Task 1b1 Core-set selection selects the most representative or useful exemplars to be stored in an episodic buffer and shared with other agents (Rios & Itti, 2018). From previous work, we expect that class-specific k-centers clustering will work well. This approach basically aims to maximally diversify which exemplars are retained, by running a clustering algorithm and only retaining one exemplar per cluster (its center), or a few.

Task 1b2 Out-of-distribution (OOD) detection. We assume that exemplars that trigger more learning are more valuable to communicate to other agents. We will explore and then select one: *Heuristic 1:* Train one epoch (one pass through the whole training set) of the new task first, then only compute the amount of learning during a subsequent epoch and rank-order exemplars accordingly. *Heuristic 2:* Compute learning for every exemplar on every epoch, then compute the average (or median, or max) learning triggered by each exemplar. *Heuristic 3:* Compute learning for every exemplar on every epoch but normalize the learning values per average within a mini-batch (or small collection of mini-batches). To summarize across epochs, we can again take the average (or median, or max) of the normalized learning values of each sample across all epochs.

Task 1b3 Dataset distillation (Wang et al., 2018) combines all training exemplars into a small number of *super-exemplars* which, when learned from using gradient descent, would generate the same gradients as the larger, original training set.

Task 1b4 Standard lossless data compression. For the purpose of transmitting curated exemplars over the network, we will investigate whether standard compression techniques (e.g., zip, bzip2, 7z) may provide a benefit. On the one hand, they may reduce network burden, but, on the other hand, they will increase computation cost (to compress and decompress the data).

Task 1b5 Lossy compression/quantization. Numerous techniques have been developed to accomplish this reduction, including pruning the network, quantizing the weights, and compressing the weights. Network pruning can reduce the number of weights necessary (LeCun et al., 1990) while maintaining performance (Malach et al., 2020) — from 9 to 13x. Weight quantization can be applied to the weights themselves — giving a further reduction of 2.3 to 3x. This can be divided into two methods. First, a straightforward reduction in the number of bits used to store the floating-point values (e.g., reducing 32 bits to 16 or 8) can be done a priori, and on some devices can also lead to a decrease in training time. Second, binning the weights into a fixed set of clusters can be done as weight sharing a priori (Chen et al., 2015) or done after training by modifying the weights to the cluster centers using uniform (Han et al., 2015) and non-uniform (Li et al., 2019) quantization. Finally, the weights can be compressed to provide an additional 1.1 to 1.5x reduction. Simple compression techniques, such as Huffman encoding, have been explored (Han et al., 2015; Lindstrom, 2014). Using all three methods together could decrease the data needed to be transmitted by 35 to 49x without performance loss (Han et al., 2015).

Task 2b: Sharing biases + curated exemplars + their logits: From previous work in knowledge distillation (Furlanello et al., 2018), training a student network with the full continuous output of the teacher (a vector of probabilities over all classes, before any final softmax), achieves better training than using a 1-hot vector that only indicates the correct class. Thus, we will investigate whether sharing a sender SKILL agent's logits along with sharing its curated exemplars can help receiving agents learn more efficiently. Network utilization will be a bit higher (to also transmit the logits; but, for each exemplar, that is only one floating-point number per output class). However, we expect that either performance will be increased, or the number of curated exemplars that need to be transmitted may be reduced to achieve same performance.

We explored the following approaches for RL:

Task 2c1 Shared Q-values pretraining. Each agent, for a given task, will learn an optimal value function (or Q-function). Subsequently, to share with other agents, we will treat this as a supervised regression problem by sharing tuples of (state, Q-values) across agents. The principles we develop in ShELL for supervised learning can be applied here to generate optimal tuples to share. Subsequently, a downstream agent will use this dataset to learn an optimal Q function for the task in consideration. This learned Q function can then be utilized to select those actions that maximize expected discounted rewards achieving the SKILL objective.

Task 2c2 Sharing of Task Specific Layers. Each agent begins with a pretrained backbone network which will behave like a feature extractor. Subsequently, each agent will learn a simple task-specific layer which uses this amortized representation to generate a Q-value, etc specific to that task. During knowledge transfer, the task specific layers will be shared across agents, allowing them to achieve high zero shot performance on target tasks.

Task 2c3 Motor Primitives/Skills as short sequences of actions that compose long-horizon complex movements. Consider for instance the task of lifting an object. This consists of many submovements like thrusting hand forward, curling fingers etc., which can be utilized for other tasks like opening a door. Thus, a set of predefined (or learnt) primitives can be provided to each of the agents and the agents can learn how to combine them for each of their respective tasks. Subsequently, during knowledge transfer, we will investigate how the low-dimensional combination representation can be shared inexpensively across agents, which informs the agent of how to best combine the primitives to succeed in the target task. A crucial advantage of this method is that receiving agents can now learn offline, without interacting with the environment, which is very fast.

Task 2c4 Hindsight Curriculum Learning whereby an agent is taught a complex long-horizon skill by dividing the task into shorter, somewhat easier subtasks, which the agent first learns to do (Portelas et al., 2020). We will train each agent on its separate tasks. Subsequently, we will utilize insights from the learning process of each agent to generate curricula for the tasks in hindsight. During the knowledge transfer phase, curricula for the tasks will be exchanged and each of the agents will use them to efficiently learn target tasks in a sample efficient manner.

Task 2c5 Offline RL Offline RL is an exciting yet fledgling field which attempts to utilize datasets of demonstrations from experts to learn an optimal policy that maximizes expected discounted rewards. Offline RL thus attempts to mitigate the problem of distribution shift, mentioned above, whereby a difference between the expert policy and the untrained policy causes an action-distribution shift leading to divergence. Recent approaches in offline RL attempt to mitigate these issues by constraining that the learnt offline policy and the expert policy are similar (measured using distance functions in the space of policies) (Nair et al., 2020; Levine et al., 2020). In our context, we will apply offline RL to allow agents to utilize expert demonstrations to learn an optimal policy without access to the actual environment from which they were generated.

Task 2d1 Graph-based approaches. Objects will be recognized as collections of visual concepts (parts) with some relative spatial configuration constraints (a graph that constrains the positions of parts). If new objects can be learned as new configurations of known parts (e.g., a semi truck has wheels like a car, but more numerous and arranged in a particular spatial configuration), then sharing knowledge can be reduced to sharing the configuration graphs, which are extremely compact. In a weaker version, SKILL agents would share both descriptions of new parts which they have encountered, and the configuration graphs. This approach is of interest here because of its very small network cost and its negligible consolidation cost at receiving agents (i.e., receiving agents

only need to add the new received graphs (and possibly parts) to their repertoires, and do not need to learn anything).

Approach	Teacher CPU (Lt+Pt)/Lt	Communication Z/D	Student CPU Ls/Lt	Speedup	Student accuracy (estimate)
Naïve	1	1	1	0.99	same
T1b1 core-set	1.001	0.009	0.009	52.60	-
T1b2 OOD	1.001	0.005	0.005	66.62	-
T1b3 data distillation	7.0269	0.002	0.0027	13.71	- -
T1b4 lossless compress	1.0120	0.9157	1.0027	0.99	same
T1b5 lossy compress	TBA<1.5	~0.2	~1	TBA	-
T2b sharing logits	1	1.001	1	0.99	+ +
T2c1 RL share Q	1	0.001	0.25	3.88	+ +
T2c2 RL share layers	1	9.96E-09	0	100.00	same
T2c3 RL motor primitives	1.001	0.001	0.001	90.83	-
T2c4 RL curriculum	TBA	TBA	TBA	TBA	TBA
T2c5 offline RL	1	0.001	0.25	3.88	+ + +
T2d1 graph-based	1.4	0.00023	0	71.42	+

Figure 11: For 100 agents, a speedup of at least $50\times$ is required in this project (green approaches). We finally decided to share the last layer + BB as an approach that achieves near perfect parallelization speedup (see main paper).

A.3 Dataset subsampling details

For dataset sampling, the following rule is used:

- For Reptilia, Fungi, Insecta, and CelebA, since it contains a lot of classes, 300 sub-classes are randomly sampled.
- For all other tasks, all classes are kept.
- For tasks with $c \geq 60$ classes, round(5120/c) train images and round(512/c) test images are used for each class. If a class does not contain enough images, then all images for that class are used.
- For tasks with $c < 60$, round(2560/c) train images and round(256/c) test images are used for each class. If a class does not contain enough images, then all images for that class are used.

A.4 GMMC number of clusters

# of clusters	2	5	10	15	20	25	30	35	40
Accuracy	74.89%	76.98%	78.27%	82.06%	82.08%	82.38%	82.24%	81.79%	81.77%

Figure 12: On a small subset of tasks, we found that $k = 25$ GMMC clusters provided the best compromise between generalization and overfitting.

A.5 Mahalanobis training MACs

The slope of MACs/image is higher until the number of training samples reaches 4,000. After that, the slope does not change. If we use 5 images per class to train, then the number of training samples would reach 4,000 after task 12. So for the majority of the tasks, the average MACs per image for training the Mahalanobis distance is around 250k.

A.6 CPU analysis

We compute everything in terms of MACs/image processed. There are a few caveats:

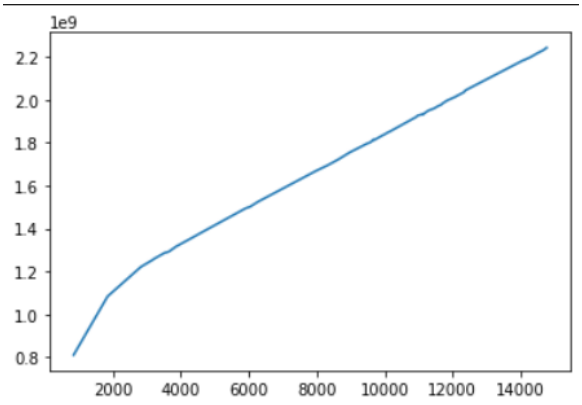


Figure 13: MACs for Mahalanobis training (vertical axis) as a function of number of training images (horizontal axis).

- Data sharing does not occur per training image, but rather per task (e.g., share 25 GMMC cluster means+diagonal covariances per task). Hence we first compute communication bytes/task and then convert that to "MACs equivalent" by assuming that sharing 1 byte takes the equivalent of 1,000 MACs.
- Mahalanobis training time increases with the number of tasks received to date. Below we assume that learning will proceed in one step using $N = 101$ agents each learning just 1 task (as opposed to each agent learning several tasks in sequence). Thus, Mahalanobis training occurs once for all 101 tasks received from all teachers, on every student agent in parallel.
- GEM training increases over time as more tasks are added:
  - We first train task 1 using the whole task 1 training set (subsampled version described above).
  - Then train task 2 using the whole task 2 training set + 10 images/class of task 1 (chosen randomly). In what follows we use γ to represent this fraction of data used for rehearsing of old tasks, and we denote by S a nominal dataset size per task (2,500 images on average). Hence, for task 2, the episodic buffer method uses $S \times (1 + \gamma)$ images. With a normalized training time of 1 to learn one task, learning task 2 for this baseline takes normalized time $1 + \gamma$.
  - Then train task 3 using the whole task 3 training set + 10 images/class of task 1 + 10 images/class of task 2. Normalized training time $1 + 2\gamma$.
  - Then train task 4 using the whole task 4 training set + 10 images/class of task 1 + 10 images/class of task 2 + 10 images/class of task 3. Normalized training time $1 + 3\gamma$.
  - etc. So the total normalized training time for N tasks is $(1)+(1+\gamma)+(1+2\gamma)+(1+3\gamma)+...+(1+(N-1)\gamma) = N+\gamma(1+2+...+N-1) = N+\gamma(N-1)(N-2)/2$. With $N = 101$, the total training time for all tasks is $N + 4999.5\gamma$. In our experiments, our subsampled training sets averaged 254 images/class and hence $\gamma = 10/254 = 0.04$ on average, leading to a total normalized training time of 299 (broken down as a cost of 101 to learn the from 101 datasets, plus 198 to rehearse old tasks as we learn new tasks).
  - This is for $\gamma = 0.04$ but performance is low, so using a higher γ is warranted for the episodic buffer approach. This is very costly, though. In the limit of retaining all images, which would give best performance, the training time of this approach is $101 + 4999.5 = 5100.5$ times the time it takes to learn one task. So, while the single-agent will require $5100.5 \times T$ to learn 101 tasks sequentially, our approach will learn all 101 tasks in parallel during just T.

  Additional details used for our computations are in Fig. 14.
- Some baselines including top-performer DERPP capped the episodic buffer to at most 10,000 images instead of growing it at a rate of 10 images/class. That is, after task 1, the

Teacher	Student	Runtime	
7.58E+11	0	8.44E+09	unfrozen xception MACs/image
0	0	8.44E+09	frozen backbone MACs/image
1.15E+07	0	9.61E+04	last layer only MACs/image
1.08E+09	0	3.58E+07	BPN only MACs/image
4.96E+06	0	8.55E+07	GMMC MACs/image
2.50E+05	250000	6.07E+08	Mahalanobis MACs/image
7.84E+11	0	8.44E+09	EWC MACs/image
3.07E+11	0	3.42E+09	PSP MACs/image
7.58E+11	0	8.44E+09	GEM MACs/image

(include all epochs)

Parameters		
# tasks (=N)	102	
MACs/byte transmitted	1000	(to equate CPU to comms)
bytes/parameter	4	
bytes/image	268203	
# xception weights	20884814	# of parameters?
xception latent dims	2048	
xception forward MACs	8.44E+09	
# GMMC clusters	25	
GMMC params/cluster	4096	
avg classes/task	46.91	
median classes/task	12.50	
avg train img/task	2577.46	
avg test img/task	269.85	
Mahalanobis img/class	5	
# BB biases	50312	
# training epochs (avg)	30	
# train images	262901	
# test images	27757	

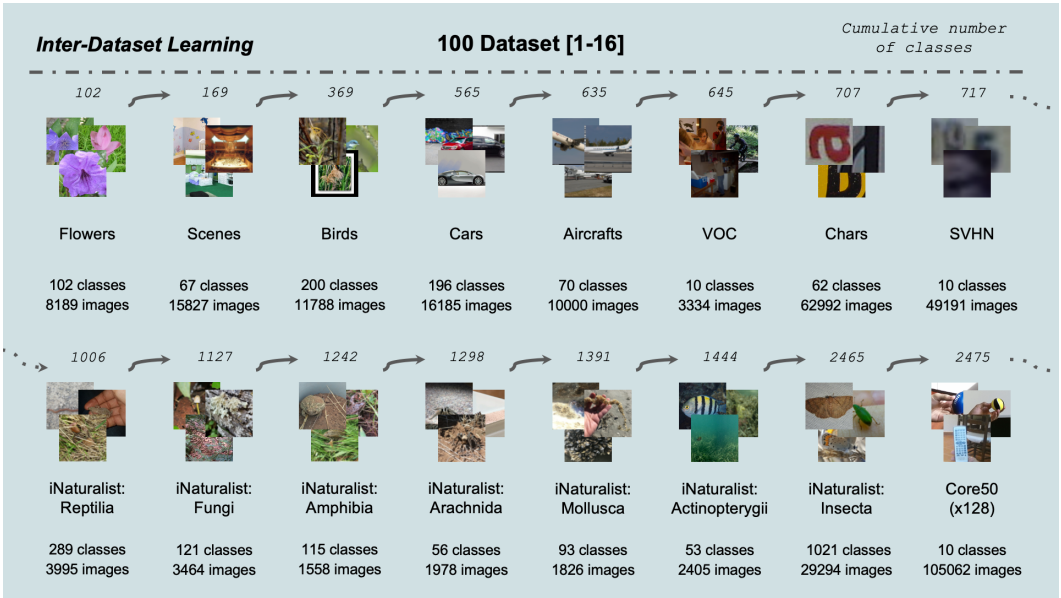
Computed from parameters	
GMMC shared bytes/task	409600
last layer bytes/task	384489
BPN biases bytes/task	201248
Mahalanobis bytes/task	62909380.1

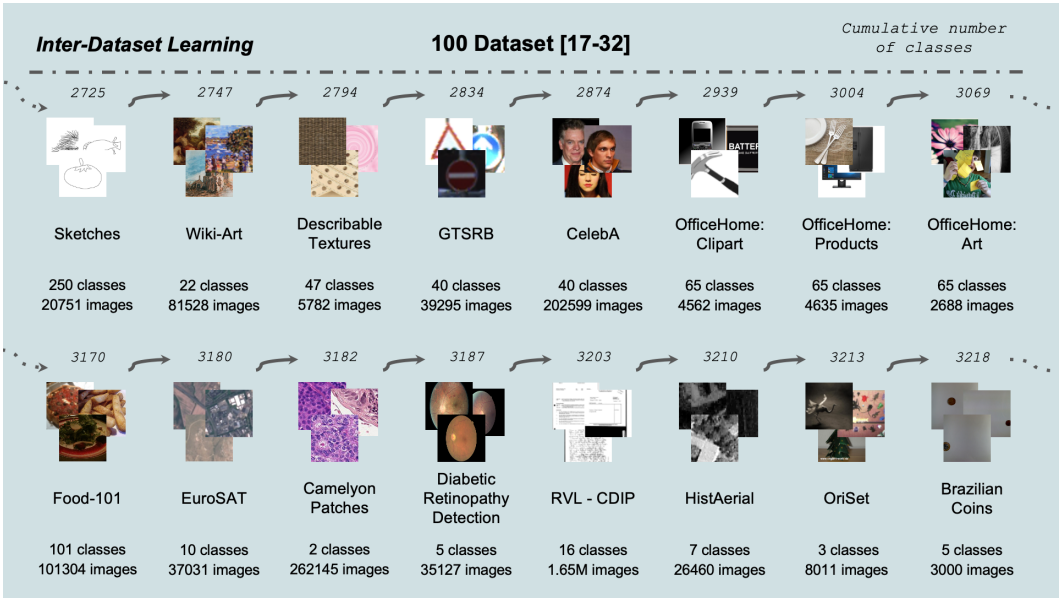
Figure 14: Additional details for how we compute MACs and speedup.

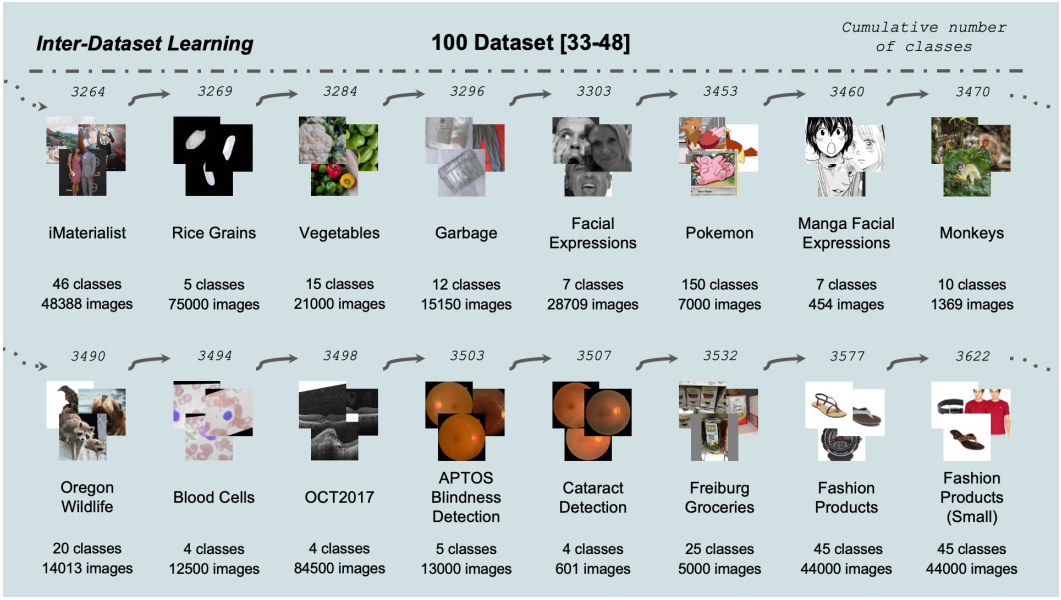
buffer will contain the whole task 1 training set (4,552 images). Training task 2 will learn from the task 2 training set + the buffer. Thus, after task 2, the buffer will contain 4,552 (from task 1) + 5,092 (from task 2) = 9,644 images. For subsequent tasks, some selection method is then deployed, by which some images from older tasks will be removed from the buffer as images from new tasks are inserted, such that total buffer size does not exceed 10,000 images. Since our training sets averaged $262,901/102 = 2,577.5$ images/task, learning task $i > 2$ takes on average $(2,577.5 + 10,000)/2,577.5 = 4.88\times$ more time than if the replay buffer was not used (naive SGD that would catastrophically fail). For 102 tasks, the total learning time is hence $\sim 498\times$ the average time to learn a single task using plain SGD.

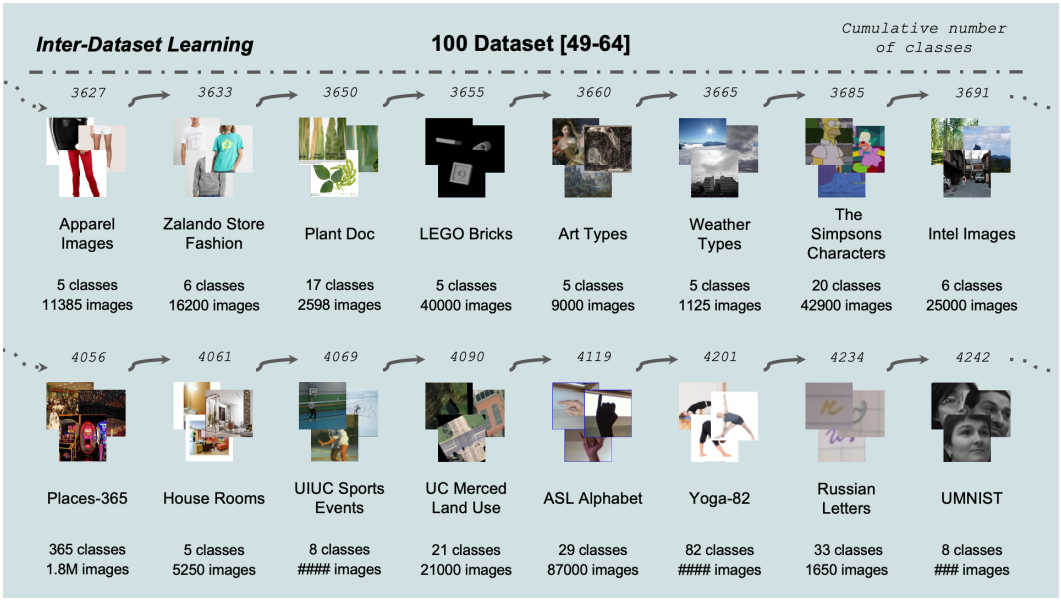
A.7 Summary of our new SKILL-102 for image classification

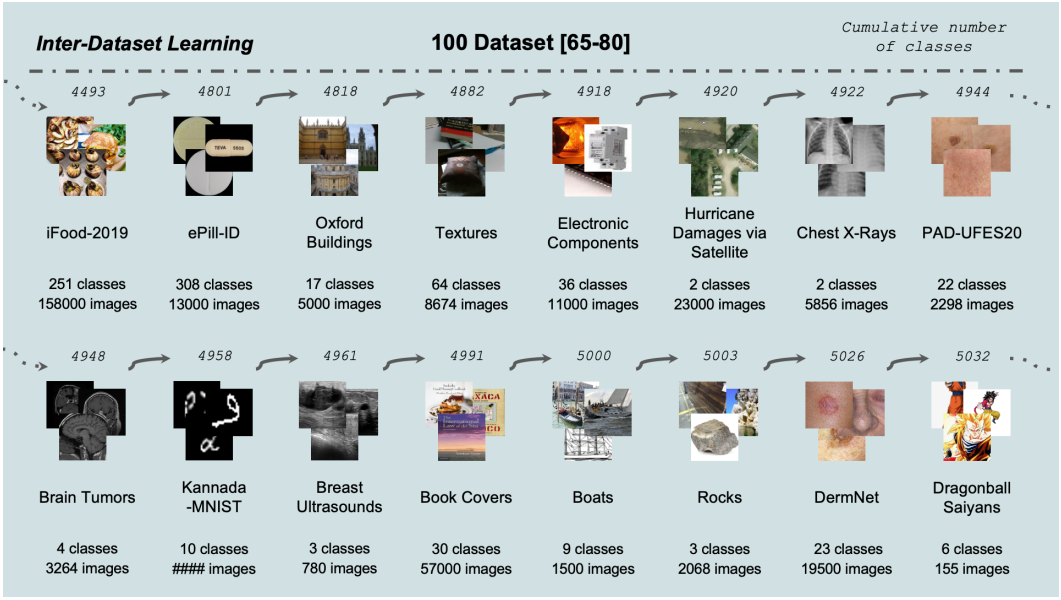
NOTE FOR REVIEW: This summary is slightly outdated, 4 datasets have changed. We will update by the time of the camera-ready version.

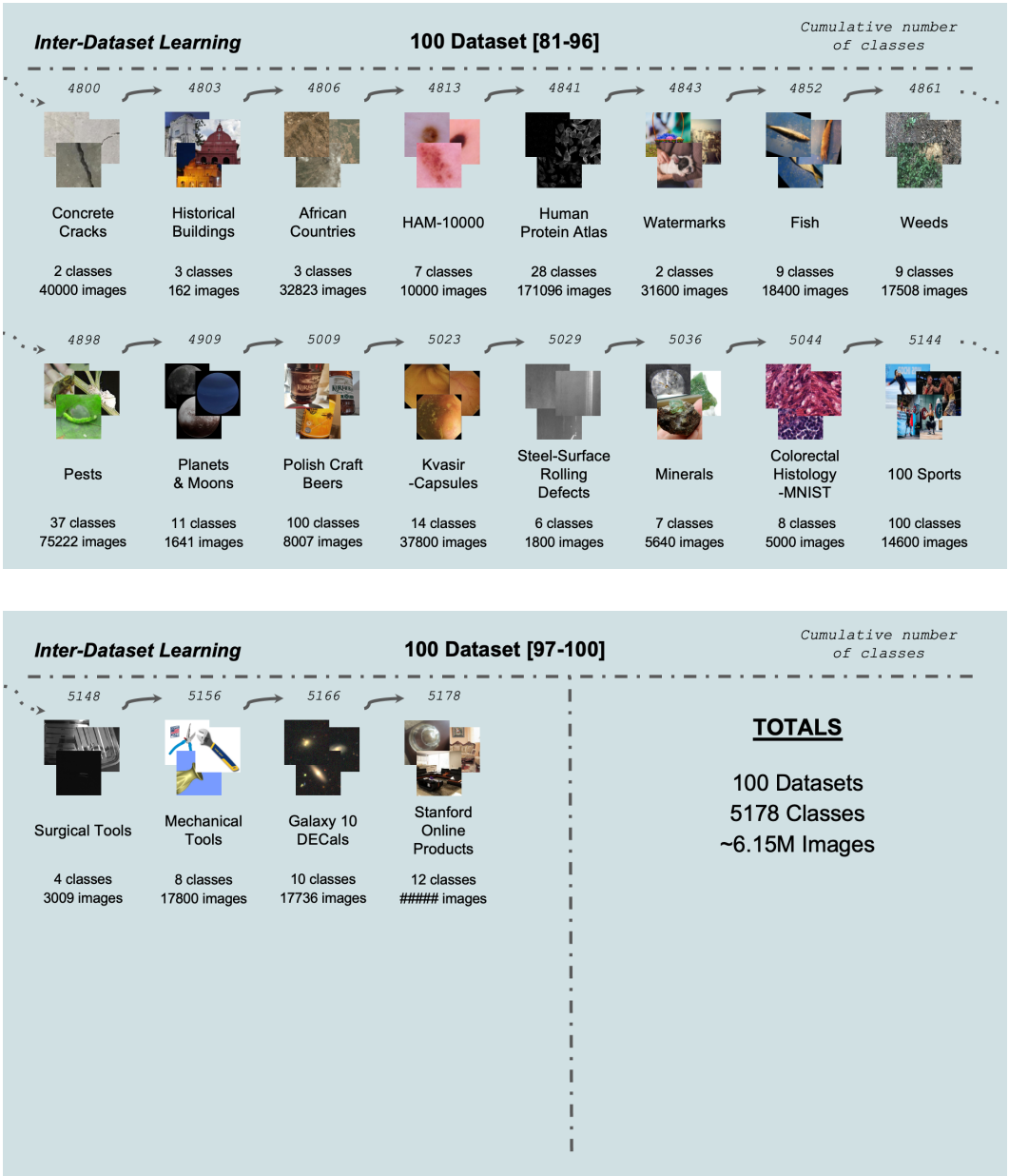



Inter-Dataset Learning
100 Dataset [17-32]
Cumulative number of classes
2725
2747
2794
2834
2874
2939
3004
3069
Sketches
250 classes
20751 images
Wiki-Art
22 classes
81528 images
Describable Textures
47 classes
5782 images
GTSRB
40 classes
39295 images
CelebA
40 classes
202599 images
OfficeHome: Clipart
65 classes
4562 images
OfficeHome: Products
65 classes
4635 images
OfficeHome: Art
65 classes
2688 images
3170
3180
3182
3187
3203
3210
3213
3218
Food-101
101 classes
101304 images
EuroSAT
10 classes
37031 images
Camelyon Patches
2 classes
262145 images
Diabetic Retinopathy Detection
5 classes
35127 images
RVL - CDIP
16 classes
1.65M images
HistAerial
7 classes
26460 images
OriSet
3 classes
8011 images
Brazilian Coins
5 classes
3000 images


Inter-Dataset Learning
100 Dataset [33-48]
Cumulative number of classes
3264
3269
3284
3296
3303
3453
3460
3470
iMaterialist
46 classes
48388 images
Rice Grains
5 classes
75000 images
Vegetables
15 classes
21000 images
Garbage
12 classes
15150 images
Facial Expressions
7 classes
28709 images
Pokemon
150 classes
7000 images
Manga Facial Expressions
7 classes
454 images
Monkeys
10 classes
1369 images
3490
3494
3498
3503
3507
3532
3577
3622
Oregon Wildlife
20 classes
14013 images
Blood Cells
4 classes
12500 images
OCT2017
4 classes
84500 images
APTOS Blindness Detection
5 classes
13000 images
Cataract Detection
4 classes
601 images
Freiburg Groceries
25 classes
5000 images
Fashion Products
45 classes
44000 images
Fashion Products (Small)
45 classes
44000 images


Inter-Dataset Learning
100 Dataset [49-64]
Cumulative number of classes
3627
3633
3650
3655
3660
3665
3685
3691
Apparel Images
5 classes
11385 images
Zalando Store Fashion
6 classes
16200 images
Plant Doc
17 classes
2598 images
LEGO Bricks
5 classes
40000 images
Art Types
5 classes
9000 images
Weather Types
5 classes
1125 images
The Simpsons Characters
20 classes
42900 images
Intel Images
6 classes
25000 images
4056
4061
4069
4090
4119
4201
4234
4242
Places-365
365 classes
1.8M images
House Rooms
5 classes
5250 images
UIUC Sports Events
8 classes
images
UC Merced Land Use
21 classes
21000 images
ASL Alphabet
29 classes
87000 images
Yoga-82
82 classes
images
Russian Letters
33 classes
1650 images
UMNIST
8 classes
images


Inter-Dataset Learning
100 Dataset [65-80]
Cumulative number of classes
4493
4801
4818
4882
4918
4920
4922
4944
iFood-2019
251 classes
158000 images
ePill-ID
308 classes
13000 images
Oxford Buildings
17 classes
5000 images
Textures
64 classes
8674 images
Electronic Components
36 classes
11000 images
Hurricane Damages via Satellite
2 classes
23000 images
Chest X-Rays
2 classes
5856 images
PAD-UFES20
22 classes
2298 images
4948
4958
4961
4991
5000
5003
5026
5032
Brain Tumors
4 classes
3264 images
Kannada -MNIST
10 classes
images
Breast Ultrasounds
3 classes
780 images
Book Covers
30 classes
57000 images
Boats
9 classes
1500 images
Rocks
3 classes
2068 images
DermNet
23 classes
19500 images
Dragonball Saiyans
6 classes
155 images



A.8 SHARING TASK INFORMATION TO PROMOTE LIFELONG LEARNING

The analysis below includes 2 options not exercised in the main text of this paper:

- **Head2Toe:** If the input domain encountered by an agent is very different than what the frozen backbone was trained on, sharing only the last layer(s) + BPN biases may not always work well, because the features in the backbone are not able to well represent the new domain. Our backbone is pretrained on ImageNet, which is appropriate for many image classification and visually-guided RL tasks in the natural world. However, the latent features may not be well suited for highly artificial worlds. This was recently addressed by (Evci et al., 2022), who showed that this problem can be alleviated using a last layer that connects to several intermediary layers, or even to every layer in the network, as opposed to only the penultimate layer. Hence, instead of sharing the last layer, we may share a so-called *Head2toe* layer when a large domain shift is encountered. Note that AR will also be used in this case as it is another way to counter large domain shifts: the AR pattern essentially recasts an input from a very different domain back into the ImageNet domain,

then allowing the frozen backbone to extract rich and meaningful features in that domain. Also see Parisi et al. (2022) for ideas similar to Head2toe, with applications in RL. Unfortunately, the CPU cost of this approach is prohibitive with respect to $0.5N$ speedup.

- **Adversarial reprogramming (AR) (Elsayed et al., 2018):** Adversarial reprogramming is quite similar in spirit to BB, with the main difference being that it operates in the input (image) space as opposed to BB operating in the activation space. In adversarial reprogramming, one computes a single noise pattern for each task. This pattern is then added to inputs for a new task and fed through the original network. The original network processes the combined input + noise and generates an output, which is then remapped onto the desired output domain. Unfortunately, the CPU cost of this approach is prohibitive with respect to $0.5N$ speedup.

We denote the number of BB biases by $\mathcal{N}$ in what follows (for xception, $\mathcal{N} = 17,472$). If Head2toe connects to the same feature maps as BB, then the number of weights is $\mathcal{N} \times c$ for c output classes. We assume that each task is modeled with k GMMC clusters ($k = 25$ currently), and each is represented by a 2048D mean and 2048D diagonal covariance. We denote by $\mathsf{4}$ the number of bytes per floating point number.

A.8.1 For a classification task with c classes:

An agent receives an image as input and produces a vector of c output values, where the highest output value is the most likely image class for the input image.

Shared params and data	Size (bytes)	Implemented: $\mathcal{N} = 17,472, c = 10, k = 25$
Last layer weights	$2048 \times c \times \mathsf{4}$	82 KBytes
BB biases	$\mathcal{N} \times \mathsf{4}$	70 KBytes
GMMC clusters	$k \times (2048 + 2048) \times \mathsf{4}$	409 KBytes
Optional: Head2toe	adds $\mathcal{N} \times c \times \mathsf{4}$	adds 699 KBytes
Optional: AR pattern	adds $299 \times 299 \times 3$	adds 268 KBytes

Total sharing per task in our current implementation: 82+70+409 = 561 KBytes/task.

When using Mahalanobis, 409 KBytes of GMMC clusters is replaced by 5 input images, which is $5 \times 299 \times 299 \times 3 = 1.34$ MByte, for a total of 1.49 MByte/task shared.

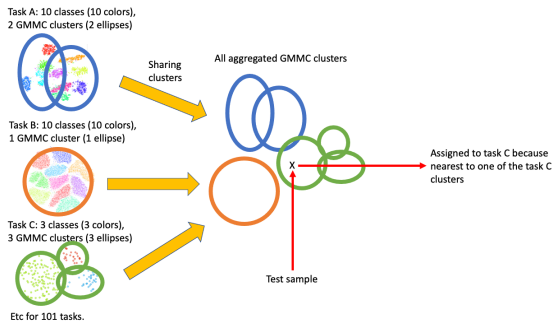
A.8.2 For a visually-guided RL task with m possible actions:

An RL agent receives video frames as input, and then selects, for each frame, one of m possible actions. This scenario fits the Atari games RL setup where inputs are screenshots from the game and actions are joystick movements. Previously, we considered a 3-layer head, but for the current experiments one layer was sufficient.

Shared params and data	Size (bytes)	Implemented: $\mathcal{N} = 17,472, m = 5$
Last layer weights	$2048 \times m \times \mathsf{4}$	40 KBytes
BB biases	$\mathcal{N} \times \mathsf{4}$	70 KBytes
GMMC clusters	$k \times (2048 + 2048) \times \mathsf{4}$	409 KBytes
Optional: Head2toe	adds $\mathcal{N} \times m \times \mathsf{4}$	adds 349 KBytes
Optional: AR pattern	adds $299 \times 299 \times 3$	adds 268 KBytes

Total sharing per task in our current implementation: 40+70+409 = 519 KBytes/task.

A.9 GMMC VISUAL EXPLANATION



GMMC task mapper. (left) Each teacher clusters its entire training set into a number of Gaussian clusters. Here, a variable number of clusters is shown for each task, but in our results below we use 25 clusters for every task. Each teacher then shares the mean and diagonal covariance of its clusters with all students. (right) Students just aggregate all received clusters in a bank, keeping track of which task any given cluster comes from. At test time, a sample is evaluated against all clusters received so far, and the task associated with the cluster closest to the test sample is chosen.

A.10 DEALING WITH OVERLAPPING TASKS

When tasks overlap, the task mapper may become confused. Here we present some discussion on this issue. Overlap *per se* is not necessarily a problem for SKILL, as long as the testing and scoring metrics can correctly establish a set of equivalence classes among overlapping labels. Consider the following example:

- Agent 1 learns a flower classification task A which, among others, contains a class for ”daisy”. Agent 2 learns task B which contains a class *”Bellis perennis”*.
- Because *Bellis perennis* is the Latin name for common daisy, it is expected that there will be overlap among the GMMC clusters for tasks A and B, as the training images for these two classes will likely look very similar.
- We believe that this is acceptable in our framework, as long as the testing and scoring mechanisms can correctly handle it. For example, consider a test image that comes from test set A and has a ground truth label of ”daisy”. If the task mapper assigns that image to task B, under naive scoring, we would immediately declare a failure of the task mapper and abort the processing of this image. However, that is potentially a mistake, as head B might still be able to correctly classify this image. Indeed, if one was to allow head B to run, and the output was *Bellis perennis*, this should be counted as an overall success for the SKILL system. This requires that it is somewhere known that daisy of task A is the same thing as *Bellis perennis* of task B.

We have not yet made equivalence assignments in the SKILL-102 dataset. We are in the process of recruiting paid student workers to do this work. We are also starting to establish a protocol for assigning these equivalences carefully. Indeed, generalizing over the above example, one may need to establish a full probabilistic ontology as opposed to a simple list of hard equivalences. For example, an image of a tiger may trigger the task mapper to invoke a head trained on various domestic cats, finally outputting a particular domestic cat breed, e.g., ”American shorthair” (which has stripes, somewhat like tigers). Although this would be a mistake, it is not as bad as if a head for vehicles was invoked by the task mapper, resulting in a final vehicle type as output, e.g., jeep. Thus, in our probabilistic ontology, tiger might be related to various domestic cats more closely than it is to vehicles. Thus, likely we will leverage existing, large ontologies, like WordNet (Miller, 1995) or ConceptNet (Liu & Singh, 2004), to assist with assigning graded degrees of equivalence among any pairs of classes from all datasets of all tasks. Using these ontologies, we will be able to compute a conceptual distance between tiger and American shorthair, with the expectation that it should be shorter than the distance between tiger and jeep. Final scoring may then just be the sum of all distances (exact hits will get a distance of 0, i.e., distance(tiger,tiger)=0, and distance(tiger,jeep) $>$ distance(tiger,American shorthair)). One final consideration for this work is whether we want to

also account for secondary classification results (e.g., consider the top 5 outputs as opposed to just the top 1, or even consider the full set of logits). We expect to have resolved these issues by the next milestone.

Other techniques are being investigated to deal with overlapping tasks, especially having an active negotiation between student and teacher on what should be shared vs. what is already known.