Coresets for Deep Learning

Hu Ding
USTC

May 7, 2025

Outline

Why Study Coreset for Deep Learning?

- **Data Explosion**: Modern deep learning models are trained on increasingly large datasets (e.g., ImageNet, OpenWebText), making training expensive and time-consuming.
- **Efficiency Bottleneck**: High computational and memory costs limit the scalability of deep models, especially on edge devices or under resource constraints.
- **Coreset Idea**: A *coreset* is a small, representative subset of the original dataset that preserves the model's training behavior.
- **Potential Benefits**:
  - Faster training and inference
  - Reduced storage and memory footprint
  - Improved generalization through better data selection
- **Challenge**: How to select informative and diverse samples that reflect both data distribution and learning dynamics?

Coresets

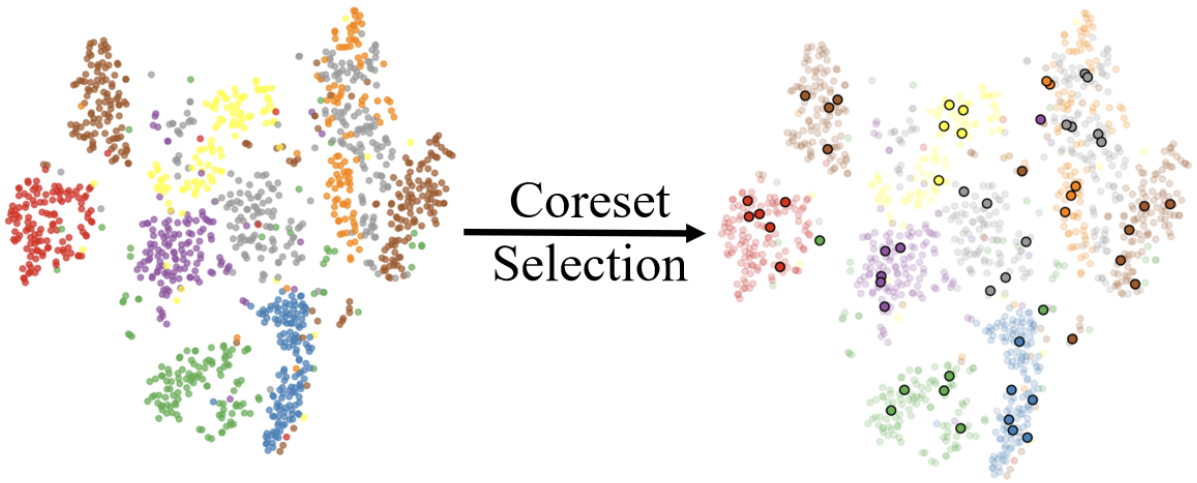


Figure: The t-SNE visualization of coreset selection.

t-SNE[1] (t-Distributed Stochastic Neighbor Embedding) is a **nonlinear** dimensionality reduction algorithm commonly used to embed high-dimensional data into 2D or 3D space for visualization. Compared to **linear methods** like **PCA**, t-SNE is better at *capturing complex nonlinear structures* such as clusters.

[1] Van der Maaten L, Hinton G. Visualizing data using t-SNE. *Journal of machine learning research*, 2008, 9(11)

Greedy Algorithms

Definition (Greedy Algorithm)

*A greedy algorithm is any algorithm that follows the problem-solving **heuristic** of making the locally optimal choice at each stage*[a].

[a] Black, Paul E. (2 February 2005). "greedy algorithm". *Dictionary of Algorithms and Data Structures*. U.S. National Institute of Standards and Technology (NIST). Retrieved 17 August 2012.

Definition (Greedy Coreset Selection)

*At each iteration (or epoch), the method appends the candidate samples that maximizes a **task-specific utility function**, thereby striving to preserve the essential statistical or geometric characteristics of the full dataset while substantially reducing its size.*

Greedy Coreset Selection in DL

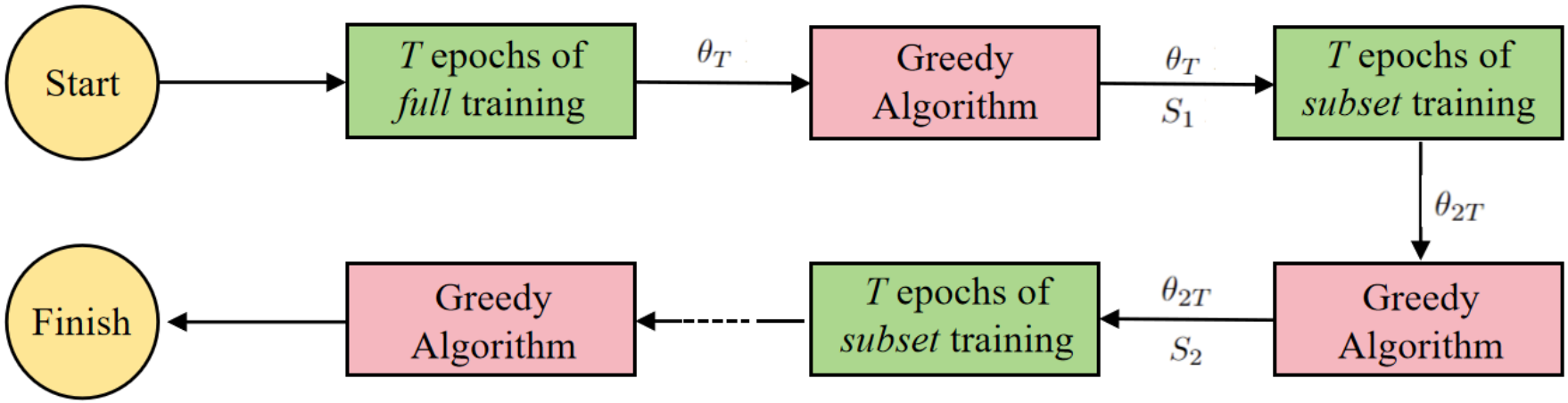


Figure: Flowchart of greedy coreset selection in DL, where coreset selection is performed every T epochs and the model is trained on the selected coreset.

1. Train on the entire dataset for T epochs (warm up).
2. Based on the parameters obtained after the T-th epoch, perform a greedy search over the dataset to select a subset S of size k (*how to select?*).
3. Train on the selected subset S for T epochs, then return to Step 2. Repeat until the algorithm converge.

Foundations of Greedy Selection

- **Dataset.**
  Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ denote a dataset, where each (x_i, y_i) corresponds to an instance and its associated label or target.

- **Set Function.**
  Consider a set function $f : 2^{\mathcal{D}} \to \mathbb{R}$ that assigns a non-negative utility score $f(S)$ to any subset $S \subseteq \mathcal{D}$. The function f is assumed to reflect task-specific objectives.

- **Objective.**
  The goal is to construct a representative subset $S \subseteq \mathcal{D}$ of fixed cardinality k, referred to as a *greedy coreset*, which approximately maximizes the utility function $f(S)$.

- **Marginal Gain.**
  At each selection step, the algorithm identifies the element $x \in \mathcal{D} \setminus S$ that maximizes the marginal gain:

$$\Delta_f(x \mid S) = f(S \cup \{x\}) - f(S),$$

  where $\Delta_f(x \mid S)$ quantifies the incremental benefit of adding x to the current subset S.

Algorithm Procedure

Algorithm Generic greedy coreset construction

Input: Dataset $\mathcal{D}$, utility function f, target size k

Initialize $S \leftarrow S_0$ ** optional warm start **/*

while $|S| < k$ **do**

  $x^\star \leftarrow \arg\max_{x \in \mathcal{D} \setminus S} \Delta_f(x \mid S)$

  $S \leftarrow S \cup \{x^\star\}$

end while

return S

Cardinality Constrained Monotone Submodular Maximization (CCMSM)

Definition (Submodular Function)

A set function $f : 2^{\mathcal{D}} \to \mathbb{R}$, defined over a finite ground set $\mathcal{D}$, is called submodular function if it satisfies

$$f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B) \tag{1}$$

for all subsets $A \subseteq B \subseteq \mathcal{D}$ and any element $x \notin B$. (diminishing returns property)

Definition (CCMSM)

The cardinality-constrained monotone submodular maximization problem can be defined as follows:

$$\max_{\mathcal{S} \subseteq \mathcal{D} : |\mathcal{S}| \leq k} f(\mathcal{S}). \tag{2}$$

Theoretical guarantee

If f is *monotone submodular function* (i.e., $f(A) \leq f(B)$ whenever $A \subseteq B$) and *normalized* (i.e., $f(\varnothing) = 0$), then the classical result by Nemhauser et al.[2] guarantees that Alg. 1 achieves the well-known $(1 - 1/e)$ approximation ratio.

Theorem (Nemhauser et al.)

Given a non-negative monotone submodular function $f : 2^{\mathcal{D}} \to \mathbb{R}$ *and the subsets* $\{S_i\}_{i \geq 0}$ *extracted by Algorithm 1, we have, for all non-negative integers* k, l*:*

$$f(S_l) \geq \left(1 - \exp\left(-\frac{l}{k}\right)\right) \max_{S:|S| \leq k} f(S), \tag{3}$$

and in particular, when $l = k$*,*

$$f(S_k) \geq \left(1 - \frac{1}{e}\right) \max_{S:|S| \leq k} f(S). \tag{4}$$

[2] Nemhauser, George L., Laurence A. Wolsey, and Marshall L. Fisher. "An analysis of approximations for maximizing submodular set functions-I." *Mathematical programming* 14 (1978): 265-294.

CRAIG

Coresets for Data-efficient Training of Machine Learning Models

Baharan Mirzasoleiman [1] **Jeff Bilmes** [2] **Jure Leskovec** [3]

Figure: [1]Department of Computer Science, University of California, Los Angeles, USA [2]Department of Electrical Engineering, University of Washington, Seattle, USA [3]Department of Computer Science, Stanford University, Stanford, USA. **Proceedings of the 37 th International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.**

Setting

Training Machine Learning Models

Often reduces to minimizing a regularized empirical risk function

Feature Label

Training data volume: $\{(x_i, y_i), i \in V\}$

Regularizer

$$w_* \in \arg\min_{w \in \mathcal{W}} f(w), \quad f(w) = \sum_{i \in V} f_i(w) + r(w), \quad f_i(w) = l(w, (x_i, y_i))$$

Loss function associated with training example $i \in V$

Examples:

- Convex $f(w)$: Linear regression, logistic regression, ridge regression, regularized support vector machines (SVM)
- Non-convex $f(w)$: Neural networks

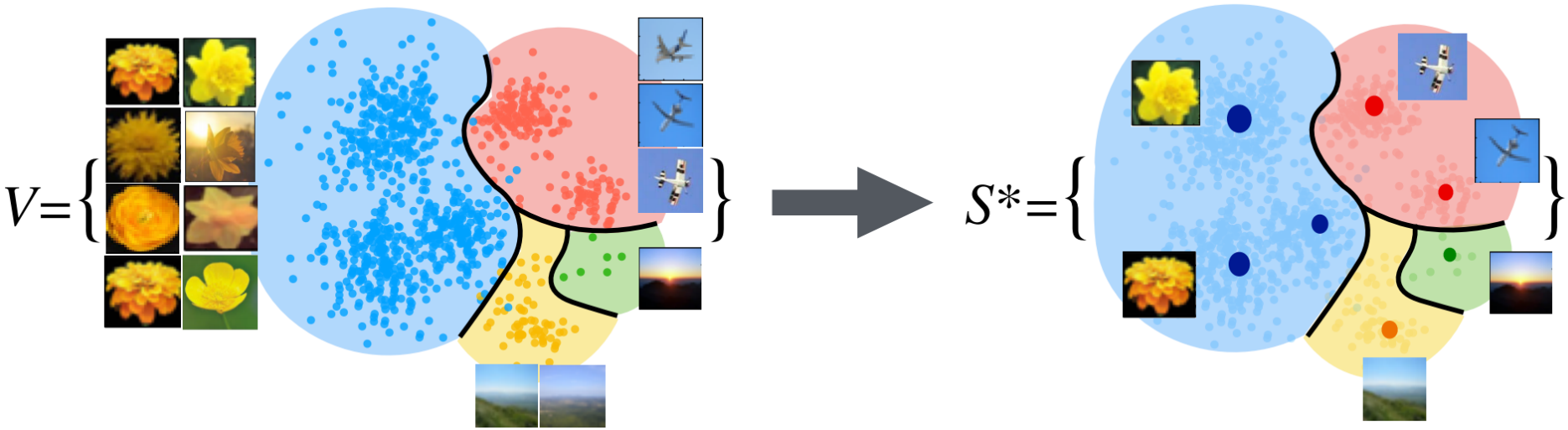
Problem

How to Find the "Right" Data for Machine Learning?

- The most informative subset

$$S^* = \arg\max_{S \subseteq V} F(S), \quad \text{s.t. } |S| \leq k$$

- What is a good choice for $F(S)$?
- If we can find S^*, we get a approximate $|V|/|S^*|$ speedup by only training on S^*

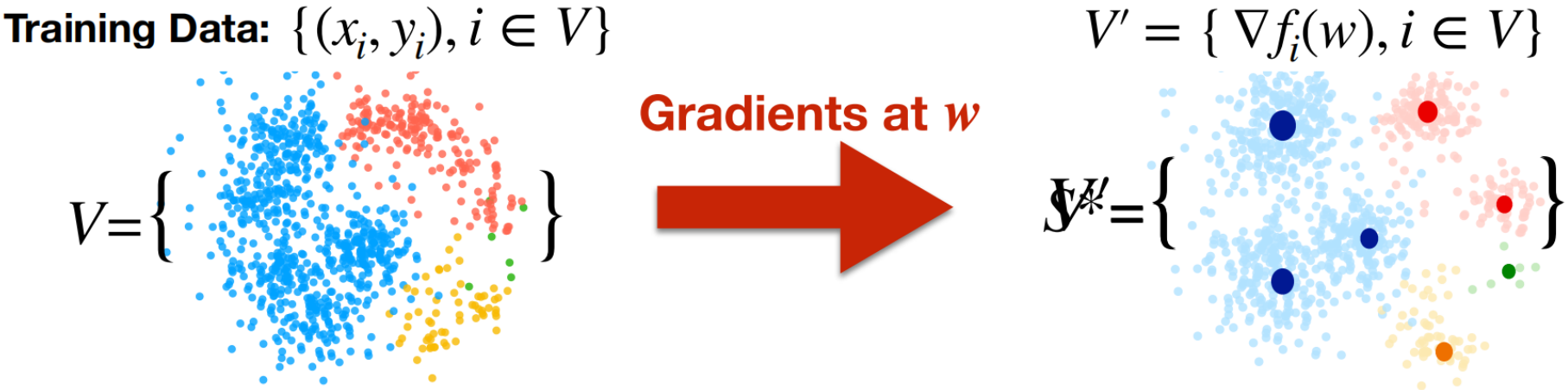


Learning from Coresets

Idea: select the smallest subset S^* and weights γ that closely estimate the full gradient.

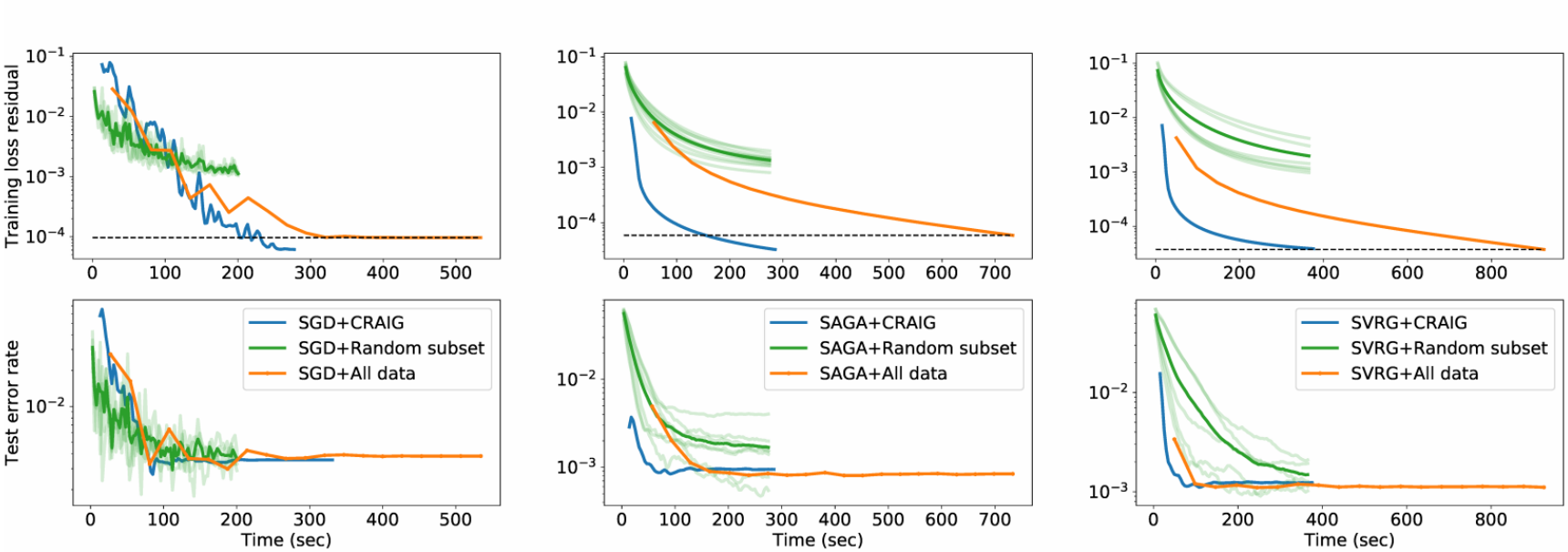
$$S^* = \arg\min_{S \subseteq V,\ \gamma_j \geq 0\ \forall j} |S|, \quad \text{s.t.} \quad \max_{w \in \mathcal{W}} \left\| \sum_{i \in V} \nabla f_i(w) - \sum_{j \in S} \gamma_j \nabla f_j(w) \right\| \leq \epsilon$$

Solution: for every $w \in \mathcal{W}$, S^* is the set of **exemplars** of all the data points in the **gradient space**.



Application of CRAIG to Logistic Regression

Training on subsets of size 10% of Covtype with 581K points



(a) SGD (b) SAGA (c) SVRG

Figure 1. Loss residual and error rate of SGD, SVRG, SAGA for Logistic Regression on Covtype data set with 581,012 data points. We compare CRAIG (10% selected subset) (blue) vs. 10% random subset (green) vs. entire data set (orange). CRAIG gives the average speedup of 3x for achieving similar loss residual and error rate across the three optimization methods.

Application of CRAIG to Neural Networks

Training on MNIST with a 2-layer neural network with 50K points

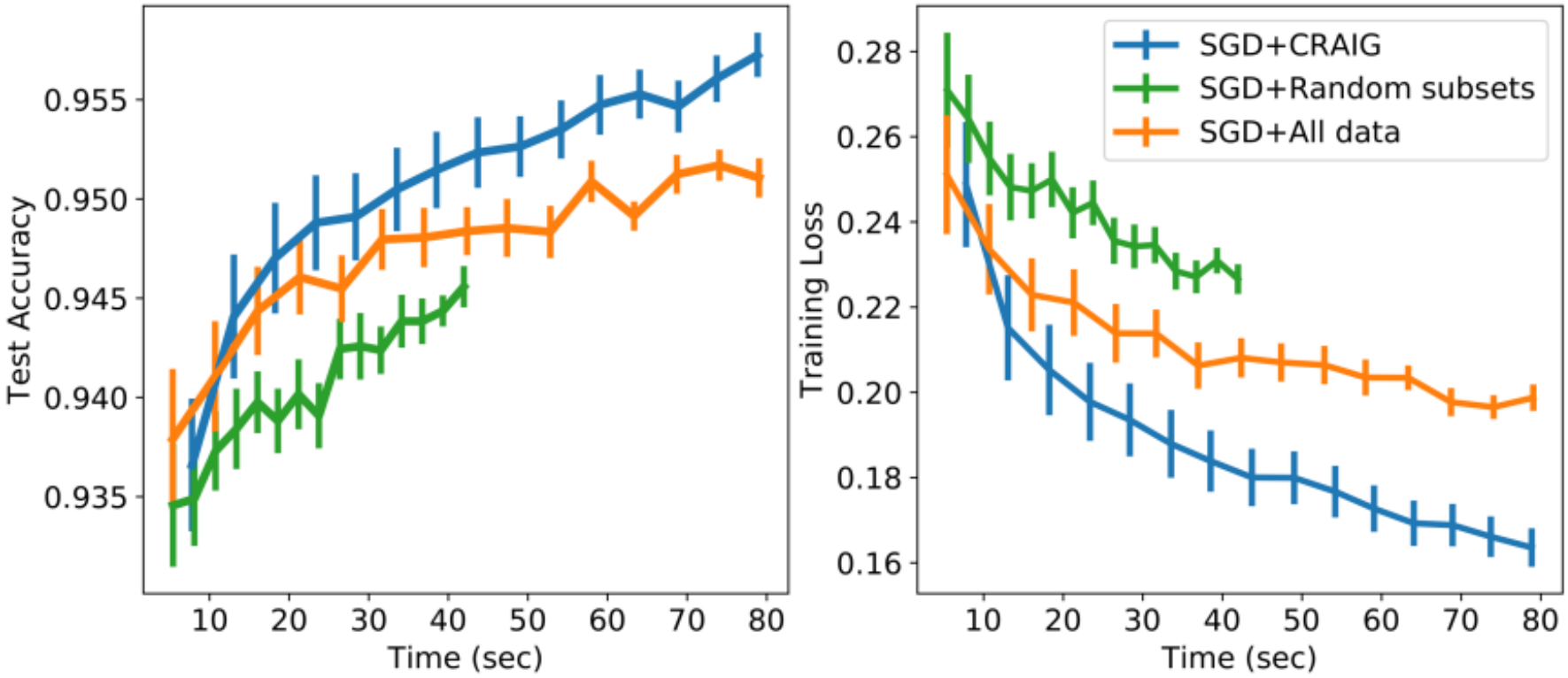


Figure: Test accuracy and training loss of SGD applied to subsets found by CRAIG vs. random subsets on MNIST with a 2-layer neural network. CRAIG provides 2x to 3x speedup and a better generalization performance.

Application of CRAIG to Deep Networks

Training ResNet20 on subsets of various size from CIFAR10 with 50K points

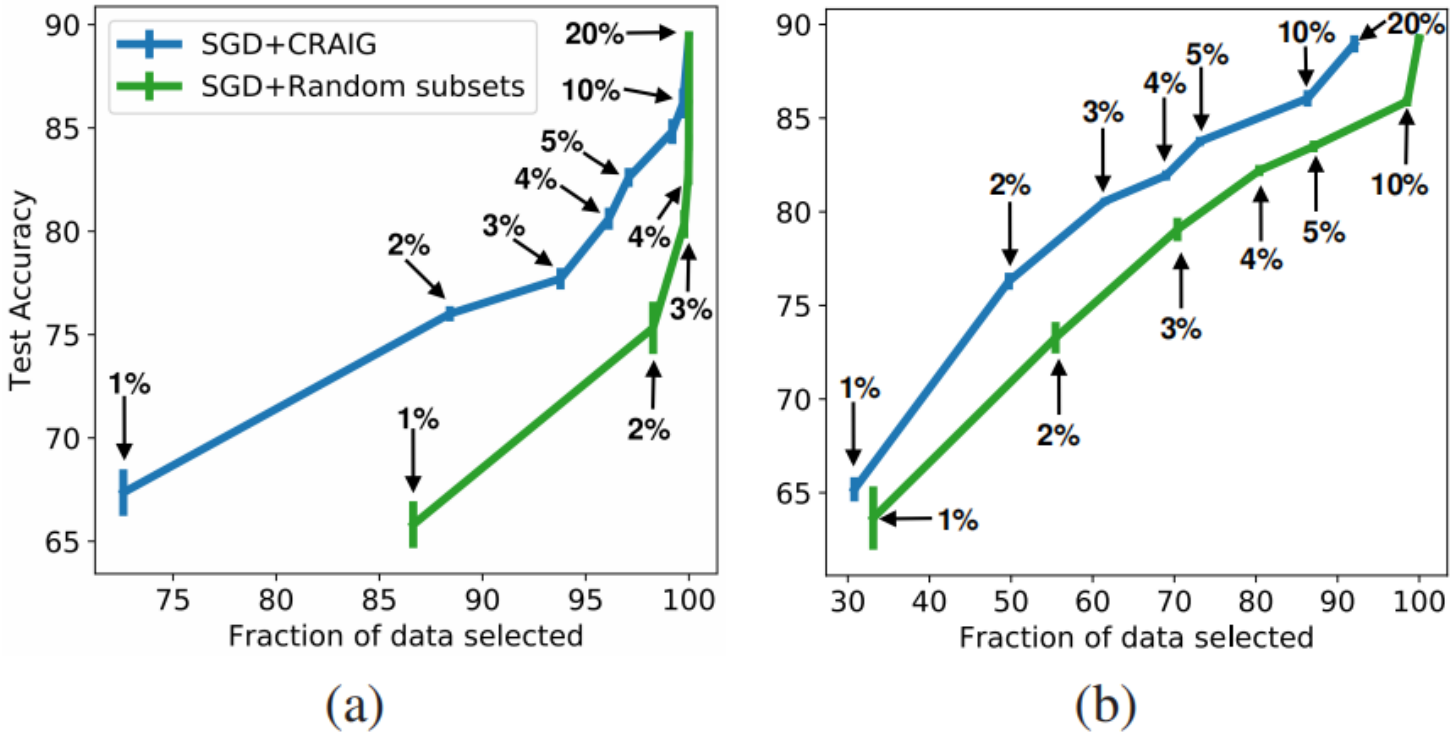
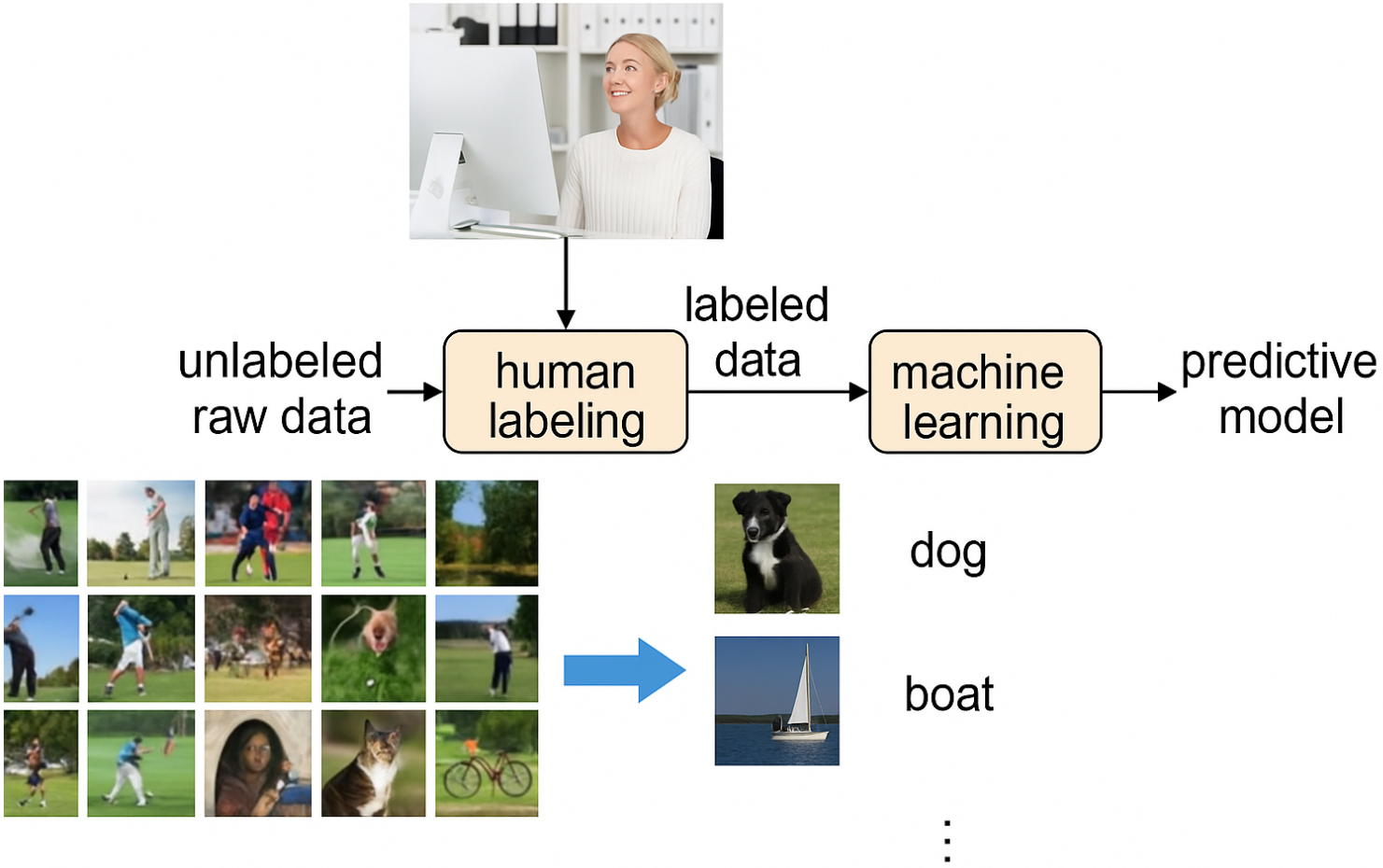


Figure: Test accuracy vs. fraction of data selected during training of ResNet20 on CIFAR10. (a) At the beginning of ever epoch, a new subset of size 1%, 2%, 3%, 4%, 5%, 10%, or 20% is selected by CRAIG. (b) Every 5 epochs a new subset of similar size is selected by CRAIG. SGD is then applied to training on the selected subsets.

Conventional (Passive) Machine Learning



Can we train machines with less labeled data and less human supervision?

Active Learning (Pool-based)

Definition

Active learning[a] *refers to the process of strategically selecting and labeling the most informative and representative samples from an unlabeled dataset to maximize learning efficiency and model accuracy.*

[a] Settles, Burr. "Active learning literature survey." (2009).

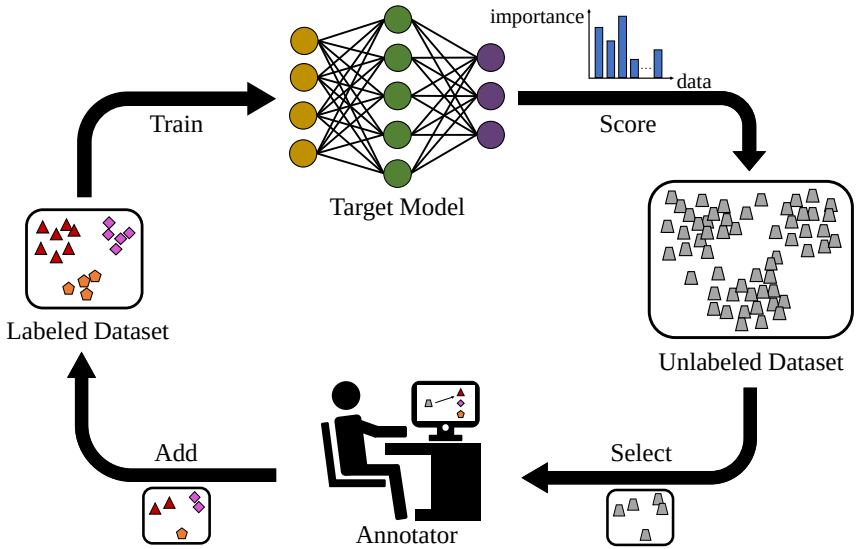


Figure: Pool-based active learning repeatedly executes four key steps: (1) training the target model using the labeled data, (2) scoring the importance of unlabeled examples and selecting the most important ones, (3) obtaining annotations for the selected examples from annotators, and (4) incorporating the newly labeled data into the existing labeled dataset.

Coreset Selection in AL

Based on the criteria for evaluating the importance of unlabeled examples, we categorize active learning approaches into three types: Loss-Based Methods (Importance); Coverage-Based Methods (Diversity); Hybrid Methods.

Type	Methods	Highlights
Loss-Based	CEAL [110]	Selecting examples using Least Confidence, Margin, or Entropy, querying true labels for some and assigning pseudo-labels to others.
	DFAL [162]	Selecting examples closest to decision boundaries.
	Yoo et al. [163]	Training a model to predict loss for unlabeled examples.
	Gao et al. [164]	Prioritizing examples with inconsistent predictions under data augmentations.
	SAAL [165]	Selecting examples exhibiting the highest training loss sharpness.
Coverage-Based	Sener et al. [114]	Formulating active learning as a k-center problem and selecting examples farthest from the already selected ones.
	DAL [117]	Employing a binary classifier to discriminate between labeled and unlabeled examples.
	VAAL [118]	Adversarially training a discriminator to discriminate between labeled and unlabeled examples.
	ProbCover [138]	Formulating active learning as a Max Probability Cover problem.
	MaxHerding [139]	Selecting examples that maximizes their similarity to the entire unlabeled dataset.
Hybrid	BADGE [119]	Selecting examples by applying k-means++ on their gradient embeddings.
	WAAL [127]	Formulating active learning as a distribution matching problem.
	NoiseStability [166]	Selecting examples based on the deviations in their feature embeddings under small perturbations.
	Cluster-Margin [120]	Performing hierarchical agglomerative clustering and selecting examples with low margin scores from the clusters.
	LDM-S [122]	Selecting high-diversity examples closest to decision boundaries.

Coresets for AL

Published as a conference paper at ICLR 2018

ACTIVE LEARNING FOR CONVOLUTIONAL NEURAL NETWORKS: A CORE-SET APPROACH

Ozan Sener*
Intel Labs
ozan.sener@intel.com

Silvio Savarese
Stanford University
ssilvio@stanford.edu

3

[3] Sener, Ozan, and Silvio Savarese. "Active Learning for Convolutional Neural Networks: A Core-Set Approach." International Conference on Learning Representations. 2018.

Coresets for AL

Coreset selection problem:

- Coreset selection problem aims to find a small subset given a large labeled dataset such that a model learned over the small subset is competitive over the whole dataset.

selects a batch of samples:

- choose ***"b"*** center points such that **the largest distance** between a data point and its nearest center is **minimized** in feature space.

$$\min_{s:|s|\leq b} \max_{i} \min_{j\in s} \Delta(\mathbf{x}_i, \mathbf{x}_j)$$

Algorithm 1 k-Center-Greedy

Input: data $\mathbf{x}_i$, existing pool $\mathbf{s}^0$ and a budget b
Initialize $\mathbf{s} = \mathbf{s}^0$
repeat
  $u = \arg\max_{i\in[n]\setminus\mathbf{s}} \min_{j\in\mathbf{s}} \Delta(\mathbf{x}_i, \mathbf{x}_j)$
  $\mathbf{s} = \mathbf{s} \cup \{u\}$
until $|\mathbf{s}| = b + |\mathbf{s}^0|$
return $\mathbf{s} \setminus \mathbf{s}^0$

Experiments

Datasets: CIFAR-10, CIFAR-100, Caltech-256, SVHN

1. The weakly-supervised model has access to labeled examples as well as unlabelled examples.
2. The fully-supervised model only has access to the labeled data points.

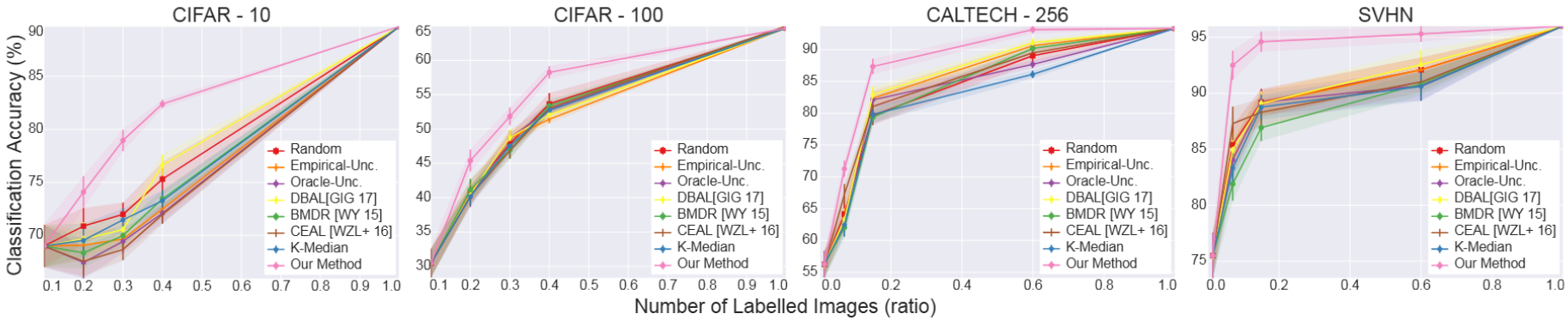


Figure 3: Results on Active Learning for Weakly-Supervised Model (error bars are std-dev)

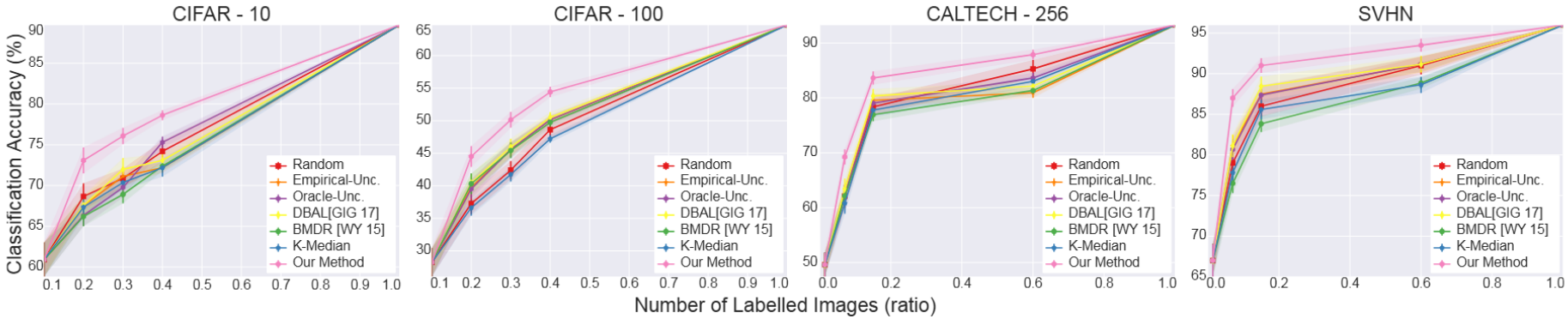
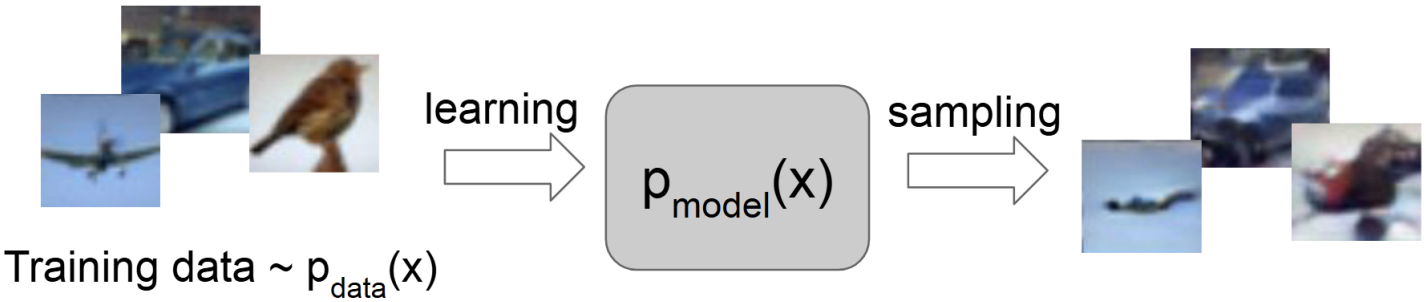


Figure 4: Results on Active Learning for Fully-Supervised Model (error bars are std-dev)

Generative Models

Given training data, generate new samples from same distribution



Objectives:

1. Learn $p_{\text{model}}(x)$ that approximates $p_{\text{data}}(x)$
2. Sampling new x from $p_{\text{model}}(x)$

Taxonomy of Generative Models

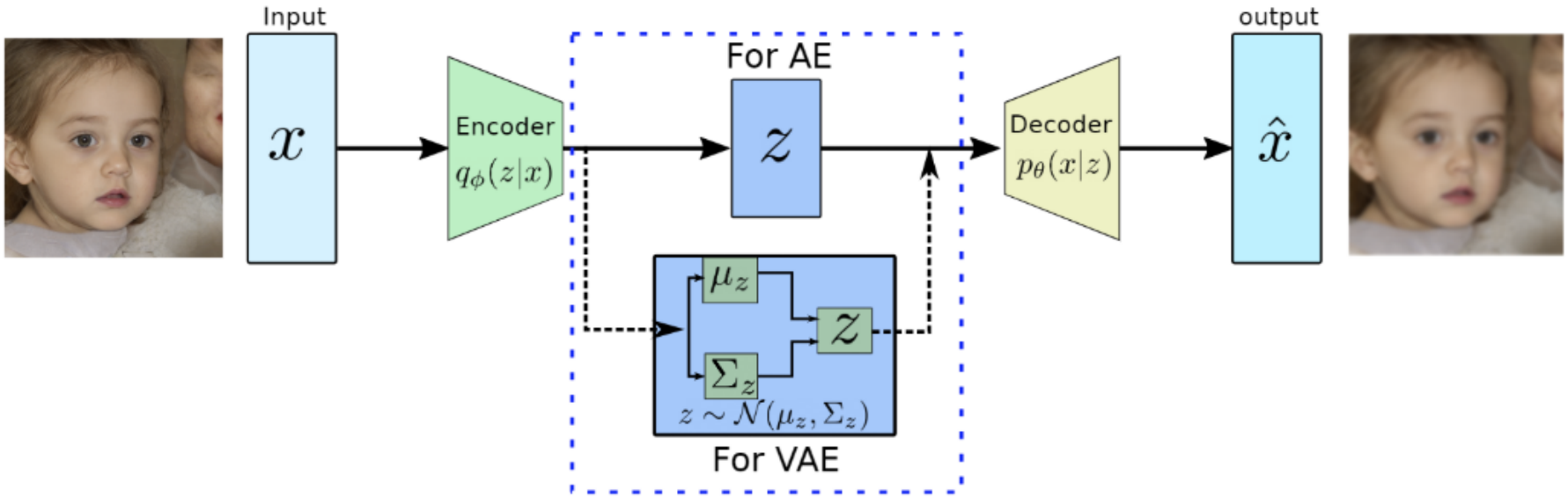
1. Generative Adversarial Networks (GAN)
2. Diffusion Models
3. Variational Autoencoders (VAE)

Coreset selection in GM

Generative Model	Methods	Highlights
GANs	Small-GAN [136]	Formulating coreset selection as a k-center problem with Euclidean distance as the distance metric.
	FDD-Coreset [137]	Formulating coreset selection as a k-center problem with Fréchet distance as the distance metric.
	Top-k GAN [108]	Discarding unrealistic synthetic images when training the generator.
	DeVries et al. [109]	Focusing on high-density regions from the data manifold.
	FIS-GAN [99]	Piroritizing challenging regions in the latent noise distribution.
VAEs	ByPE-VAE [126]	Employing Bayesian pseudocoresets to accelerate the training of Exemplar VAE.
Diffusion Models	Li et al. [142]	Coresets with learnable class-wise weights.
	Briq et al. [143]	Experimentally evaluating several coreset selection methods.

Variational Autoencoder (VAE)

- VAE: a likelihood-based generative model



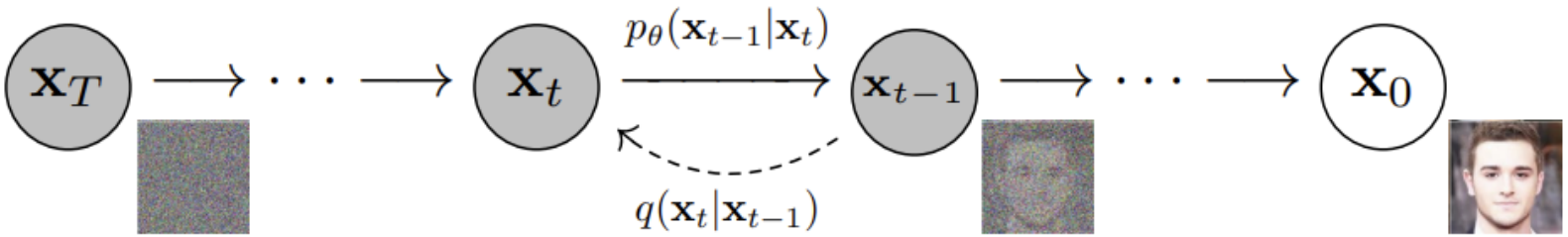
- Encoder: a model that approximates the posterior $q(z|x)$
- Decoder: a model that transforms a Gaussian variable z to real data
- Training: maximize the ELBO

$$\mathcal{L}(\theta, \phi; x) = \mathbb{E}_{z \sim q_\phi(z|x)} \left[\log p_\theta(x|z)\right] - D_{KL}\left(q_\phi(z|x) \,\|\, p_\theta(z)\right),$$

Diffusion Model

Denoising Diffusion Probabilistic Models (DDPM)[4]

1. Diffusion models have two processes
2. Forward diffusion process gradually adds noise to input
3. Reverse denoising process learns to generate data by denoising



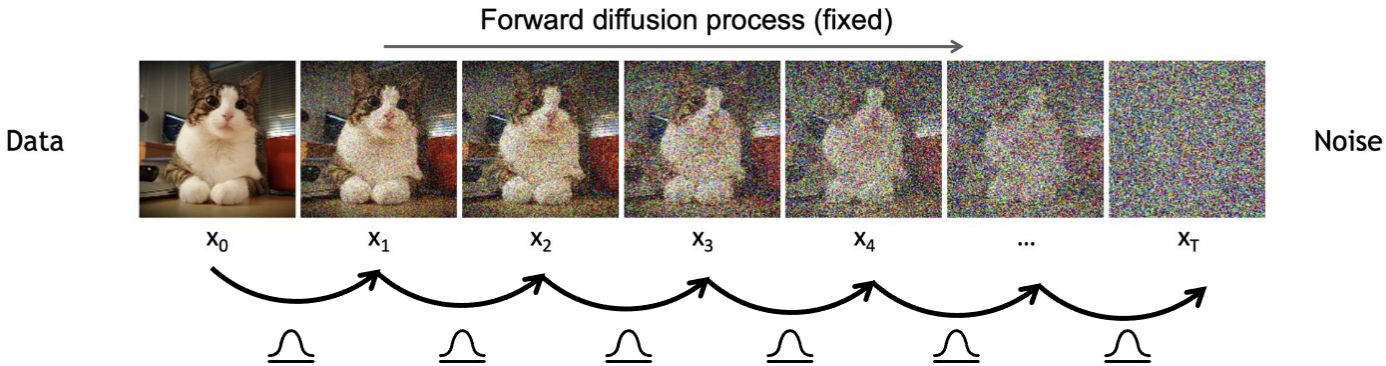
[4]Ho, Jonathan, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models." Advances in neural information processing systems 33 (2020): 6840-6851.

Forward Diffusion Process

1. Forward diffusion process is stacking **fixed VAE encoders**
   - gradually adding Gaussian noise according to schedule β_t

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}\right)$$

$$q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$$



2. The forward process allows sampling of x_t at arbitrary timestep t in closed form:

$$q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}\right) \quad \bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$$

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$$

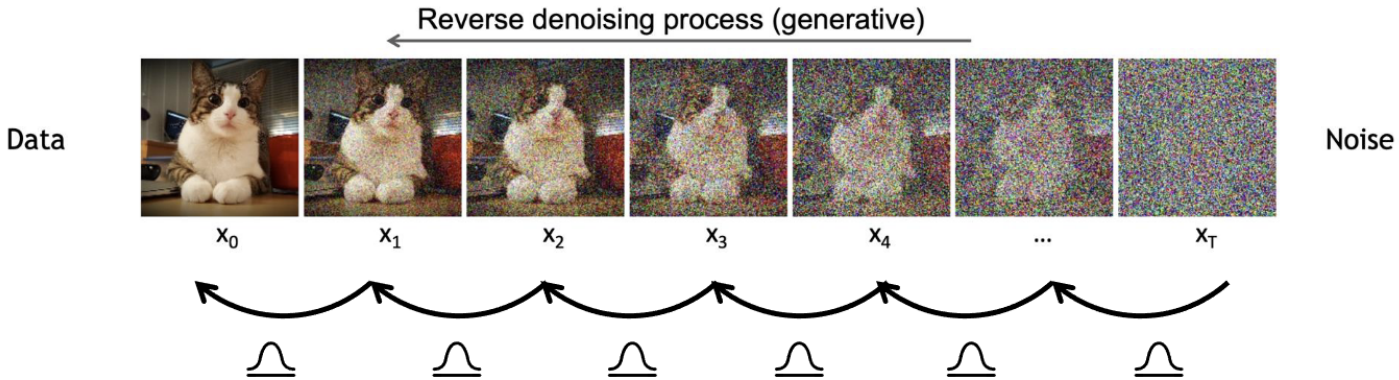
3. The noise schedule (β_t values) is designed such that $q(\mathbf{x}_T \mid \mathbf{x}_0) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$

Reverse Denoising Process

1. Generation process
   - Sample $x_t \sim \mathcal{N}(x_T; \mathbf{0}, \mathbf{I})$
   - Iteratively sample $x_{t-1} \sim q(x_{t-1}|x_t)$
2. $q(x_{t-1}|x_t)$ not directly tractable
3. But can be estimated with a Gaussian distribution if β_t is small at each step
4. Reverse diffusion process is stacking learnable VAE decoders
   - Predicting the mean and std of added Gaussian Noise

$$p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}) \quad p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \underbrace{\mu_\theta(\mathbf{x}_t, t), \sigma_t^2 \mathbf{I}}_{\text{Trainable Network, Shared Across All Timesteps}})$$



Learning the Denoising Model

1. Denoising models are trained with variational upper bound (negative ELBO), as VAEs

$$\mathbb{E}_{q(\mathbf{x}_0)}\left[-\log p_\theta(\mathbf{x}_0)\right] \leq \mathbb{E}_{q(\mathbf{x}_0)q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}\right] =: \mathcal{L}$$

2. In DDPM, this is further simplified to:

$$L = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0),\, \boldsymbol{\epsilon} \sim \mathcal{N}(0,\mathbf{I}),\, t \sim \mathcal{U}(1,T)}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\left(\underbrace{\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}}_{\mathbf{x}_t},\ t\right)\right\|^2\right]$$

3. **Summary:** Training and Sampling

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \ldots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$\nabla_\theta \left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t)\right\|^2$
6: **until** converged

Algorithm 2 Sampling

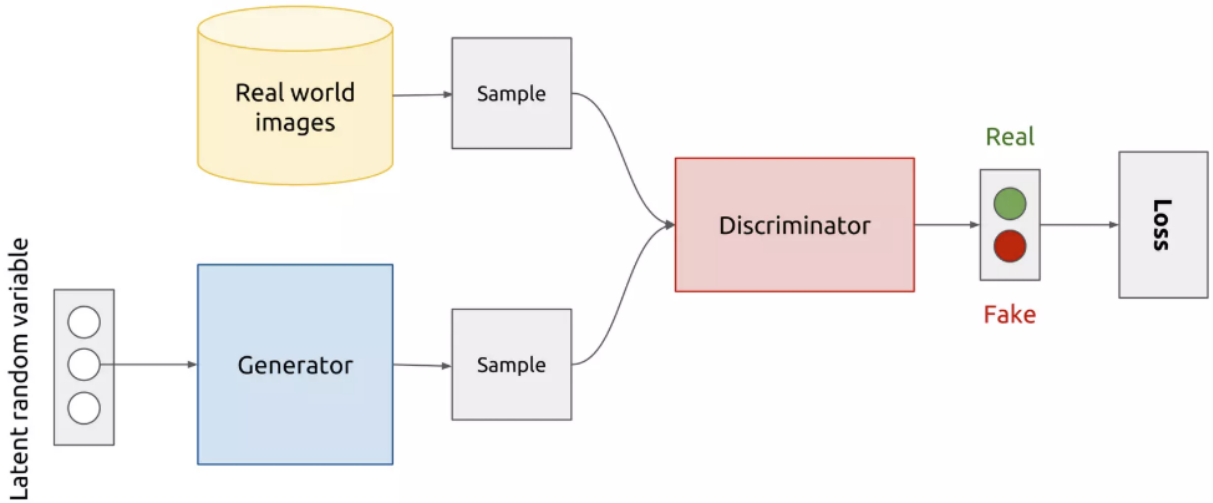
1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

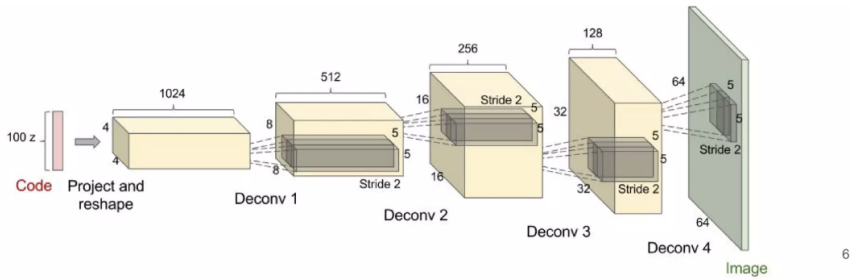
Generative Adversarial Networks (GANs)

- A method of training deep generative models
- **Idea**: A generator and a discriminator against each other
- Generator tries to draw samples from $p_{\text{model}}(x)$
- Discriminator tries to tell if sample came from the generator or the real world
- Both discriminator and generator are deep networks (differentiable functions)
- Can train with backprop: train discriminator for a while, then train generator, then discriminator, . . .

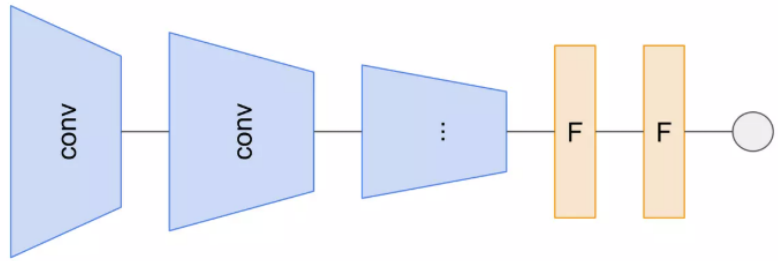
Generative Adversarial Networks (GAN)

GAN's Architecture[5]





(a) Generator (usually a DNN)



(b) Discriminator (usually a CNN)

[5] slideshare.net/slideshow/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016/64667744

Small-GAN: Speeding Up GAN Training Using Core-sets[6]

1. Training with very large mini-batches has been shown to significantly improve performance [Brock et al. (2018)].
2. But using large batches is slow and expensive.

Batch-Sizes	256	512	1024	2048
FID	18.65	15.30	14.88	12.39

How can we use large batch-size without overhead cost?

If we could generate batches that were effectively large though actually small.

[6] Sinha, Samarth, et al. "Small-gan: Speeding up gan training using core-sets." International Conference on Machine Learning. PMLR, 2020.

GreedyCoreset

Setting: In computational geometry, a Coreset Q of a set P is a subset $Q \subset P$ that approximates the 'shape' of P [Agarwal et al., 2005].

The Coreset selection problem can be formulated as the *minimax facility location* formulation:

$$\min_{Q:|Q|=k} \max_{x_i \in P} \min_{x_j \in Q} d(x_i, x_j)$$

- Exact solution to set-covers is NP-Hard
- We can use Greedy Approximations

Algorithm 1 `GreedyCoreset`

Input: batch size (k), data points (x where $|x| > k$)
Output: subset of x of size k

$s \leftarrow \{\}$ ▷ Initialize the sampled set
while $|s| < k$ **do** ▷ Iteratively add points to sampled set
  $p \leftarrow \arg\max_{x_i \notin s} \min_{x_j \in s} d(x_i, x_j)$
  $s \leftarrow s \cup \{p\}$
end whilereturn s

Small-GAN

Algorithm 2 `Small-GAN`

Input: target batch size (k), starting batch size ($n > k$), Inception embeddings (ϕ_I)
Output: a trained GAN

Initialize networks G and D
for $step = 1$ to ... **do**
- $z \sim p(z)$ ▷ Sample n points from the prior
- $x \sim p(x)$ ▷ Sample n points from the data distribution
- $\phi(x) \leftarrow \phi_I(x)$ ▷ Get cached embeddings for x
- $\hat{z} \leftarrow \texttt{GreedyCoreset}(z)$ ▷ Get Core-set of z
- $\widehat{\phi(x)} \leftarrow \texttt{GreedyCoreset}(\phi(x))$ ▷ Get Core-set of embeddings
- $\hat{x} \leftarrow \phi_I^{-1}(\widehat{\phi(x)})$ ▷ Get x corresponding to sampled embeddings
- Update GAN parameters as usual

end for

Q: Why use Inception embeddings $\phi_I(\cdot)$?

Small-GAN

Algorithm 2 `Small-GAN`

Input: target batch size (k), starting batch size ($n > k$), Inception embeddings (ϕ_I)
Output: a trained GAN

Initialize networks G and D
for $step = 1$ to ... **do**
  $z \sim p(z)$ ▷ Sample n points from the prior
  $x \sim p(x)$ ▷ Sample n points from the data distribution
  $\phi(x) \leftarrow \phi_I(x)$ ▷ Get cached embeddings for x
  $\hat{z} \leftarrow \texttt{GreedyCoreset}(z)$ ▷ Get Core-set of z
  $\widehat{\phi(x)} \leftarrow \texttt{GreedyCoreset}(\phi(x))$ ▷ Get Core-set of embeddings
  $\hat{x} \leftarrow \phi_I^{-1}(\widehat{\phi(x)})$ ▷ Get x corresponding to sampled embeddings
  Update GAN parameters as usual
end for

Q: Why use Inception embeddings $\phi_I(\cdot)$?
A: Running GreedyCoreset on images is ineffective because of noise and high dimensionality [Donoho et al., 2000]. Inception embeddings of images are cheap and effective

Experiments:CIFAR10

Small-GAN significantly outperforms regular GAN training at all batch-sizes

GAN (batch-size = 128)	Small-GAN (batch-size = 128)	GAN (batch-size = 256)	Small-GAN (batch-size = 256)	GAN (batch-size = 512)	Small-GAN (batch-size = 512)
18.75 ± 0.2	$\mathbf{16.73 \pm 0.1}$	17.9 ± 0.1	$\mathbf{16.22 \pm 0.3}$	15.68 ± 0.2	$\mathbf{15.08 \pm 0.1}$

Figure: FID scores for CIFAR using SN-GAN

Experiments:LSUN+ImageNet

Small-GAN able to scale well to large scale settings

Small-GAN (64)	GAN (64)	GAN (128)	GAN (256)
13.08	14.82	13.02	12.63

Table: FID scores using SAGAN for LSUN dataset

GAN	Small-GAN
19.40	**17.33**

Table: FID scores using SAGAN for ImageNet dataset

Continual Learning (Experience rehearsal based or Coreset based)

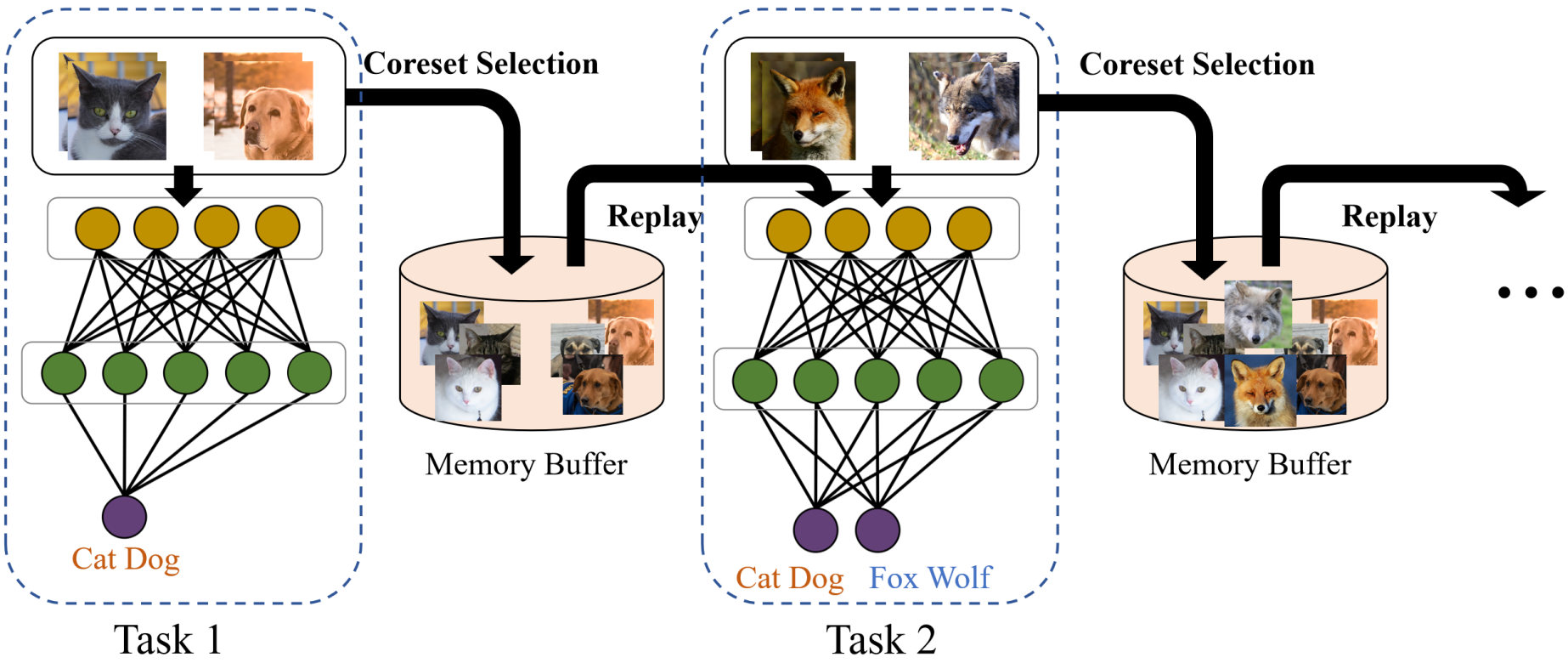


Figure: Overview of the coreset-based method for continual learning. This approach aims to approximate and recover previous data distributions by storing a small subset of past training samples in a limited *memory buffer*. By leveraging this stored subset, the method helps mitigate *catastrophic forgetting*, ensuring that the model retains knowledge from earlier learning phases while adapting to new data.

Coresets in CL

Table 3 Coreset selection methods in continual learning.

Type	Methods	Highlights
Gradient-Free Strategies	Reservoir Sampling [193, 194, 195, 196]	A fundamental streaming algorithm for maintaining fixed-size uniform samples from unbounded data streams of unknown cardinality.
	Ring Buffer [197, 198, 199]	A class-partitioned memory architecture designed to ensure balanced exemplar retention.
	Herding [200, 201]	A method that dynamically adjusts slice dimensions by allocating $\|\mathcal{M}\|/C_{seen}$ slots per class based on the number of observed classes C_{seen}.
	k-Means [134]	Use online k-Means to estimate the k class centroids in the penultimate feature space and store the input samples closest to each centroid in memory.
Gradient-Driven Strategies	GSS [202]	Formulate coreset selection as a constraint optimization problem of continual learning.
	OCS [203]	A coreset selection method that selects the most representative and informative coreset at each iteration and trains them in an online manner.
	Greedy Coreset [63]	Propose a coreset construction via cardinality-constrained bilevel optimization.
	PBCS [82]	A continuous probabilistic bilevel formulation of coreset selection by learning a probablistic weight for each training sample.
	BCSR [81]	A new bilevel formulation where the inner problem minimizes expected training error from a sampled distribution, and the outer problem learns a sparse distribution with K nonzero entries to minimize overall training error.
	GCR [192]	A method for selecting a coreset that approximates the model parameter gradients over all previously seen data.
	RM [204]	A method that emphasize the importance of maintaining sample diversity in coreset.
	ASER [205]	Propose a novel Adversarial Shapley value scoring method for coreset selection in continual learning.

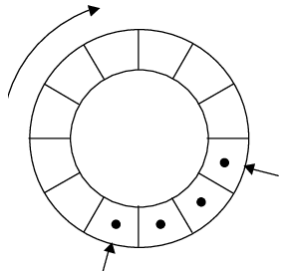
Gradient-free strategies

1. Reservoir Sampling

Algorithm 3 - Reservoir Sampling

Input: memory buffer $\mathcal{M}$, number of seen examples N, example x, label y.
if $\mathcal{M} > N$ **then**
  $\mathcal{M}[N] \leftarrow (x, y)$
else
  $j = \textit{randomInteger}\ (\min = 0, \max = N)$
  if $j < |\mathcal{M}|$ **then**
    $\mathcal{M}[j] \leftarrow (x, y)$
  end if
end if
return M

2. Ring Buffer



3. Herding
4. k-Means

Gradient-driven strategies

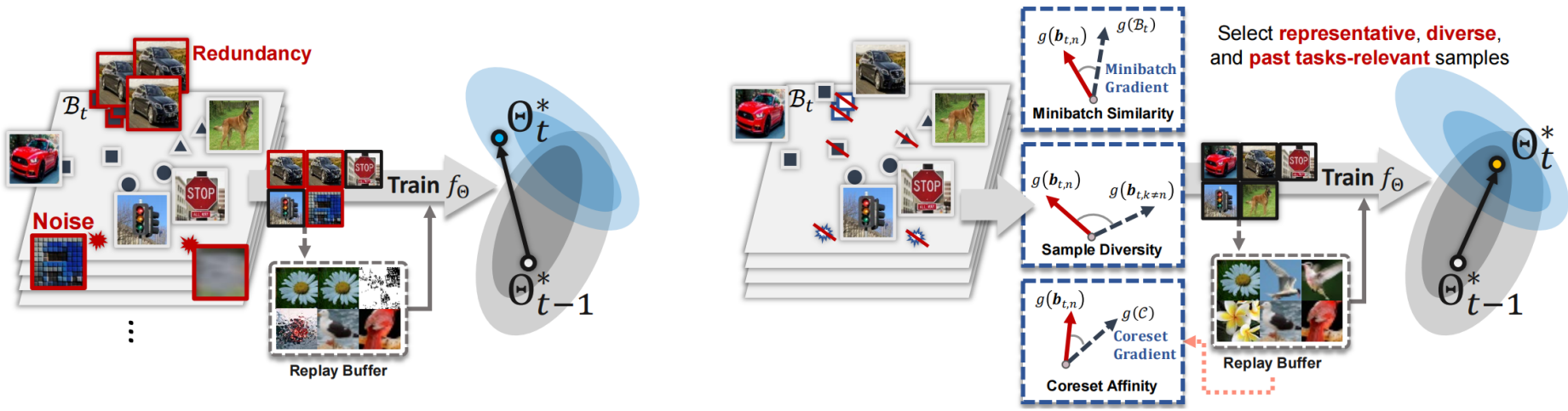
ONLINE CORESET SELECTION FOR REHEARSAL-BASED CONTINUAL LEARNING

Jaehong Yoon[1] **Divyam Madaan**[2*] **Eunho Yang**[1,3] **Sung Ju Hwang**[1,3]
KAIST[1] New York University[2] AITRICS[3]
jaehong.yoon@kaist.ac.kr, divyam.madaan@nyu.edu,
eunhoy@kaist.ac.kr, sjhwang82@kaist.ac.kr

7

[7] Yoon, Jaehong, et al. "Online Coreset Selection for Rehearsal-based Continual Learning." International Conference on Learning Representations.

OCS



(a) Existing rehearsal-based CL

(b) Online Coreset Selection (OCS)

Figure: Illustration of existing rehearsal-based CL and Online Coreset Selection (OCS).

- Existing rehearsal-based methods train on all the arrived instances and memorize a fraction of them in the replay buffer, which results in a suboptimal performance due to the outliers (noisy or biased instances).
- OCS obtains the coreset by leveraging our **three selection strategies**, which discard the outliers at each iteration.

Three Different Selection Strategies

Let $b_{t,n} = \{x_{t,n}, y_{t,n}\} \in \mathcal{B}_t$ denote the n-th pair of data point with gradient $\nabla f_\Theta(\mathbf{b}_{t,n})$ and its corresponding label at task $\mathcal{T}_t$.

1. Minibatch similarity

$$\mathcal{S}(b_{t,n} \mid \mathcal{B}_t) = \frac{\nabla f_\Theta(b_{t,n})\, \bar{\nabla} f_\Theta(\mathcal{B}_t)^\top}{\left\|\nabla f_\Theta(b_{t,n})\right\| \cdot \left\|\bar{\nabla} f_\Theta(\mathcal{B}_t)\right\|}.$$

2. Sample diversity

$$\mathcal{V}(b_{t,n} \mid \mathcal{B}_t \setminus b_{t,n}) = \frac{-1}{N_t - 1} \sum_{p \neq n}^{N_t - 1} \frac{\nabla f_\Theta(b_{t,n})\, \nabla f_\Theta(b_{t,p})^\top}{\left\|\nabla f_\Theta(b_{t,n})\right\| \cdot \left\|\nabla f_\Theta(b_{t,p})\right\|}.$$

3. Coreset affinity

$$\mathcal{A}(b_{t,n} \mid \mathcal{B}_C \sim C) = \frac{\nabla f_\Theta(b_{t,n})\, \bar{\nabla} f_\Theta(\mathcal{B}_C)^\top}{\left\|\nabla f_\Theta(b_{t,n})\right\| \cdot \left\|\bar{\nabla} f_\Theta(\mathcal{B}_C)\right\|}.$$

Algorithm

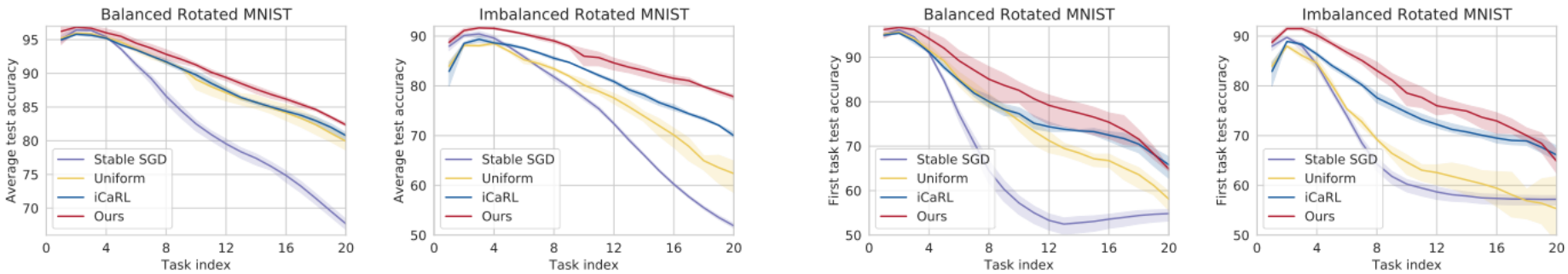
Algorithm 1 Online Coreset Selection (OCS)

input Dataset $\{\mathcal{D}_t\}_{t=1}^{T}$, neural network f_Θ, hyperparameters λ, τ, replay buffer $\mathcal{C} \leftarrow \{\}$, buffer size bound J.

for task $\mathcal{T}_t = \mathcal{T}_1, \ldots, \mathcal{T}_T$ **do**
  $\mathcal{C}_t \leftarrow \{\}$ ▷ Initialize coreset for current task
  for batch $\mathcal{B}_t \sim \mathcal{D}_t$ **do**
    $\mathcal{B}_\mathcal{C} \leftarrow \textsc{Sample}(\mathcal{C})$ ▷ Randomly sample a batch from the replay buffer
    $\mathbf{u}^* = \underset{n \in \{0,\ldots,|\mathcal{B}_t|-1\}}{\text{argmax}^{(\kappa)}} \mathcal{S}\left(\boldsymbol{b}_{t,n} \mid \mathcal{B}_t\right) + \mathcal{V}\left(\boldsymbol{b}_{t,n} \mid \mathcal{B}_{t \setminus b_{t,n}}\right) + \tau \mathcal{A}\left(\boldsymbol{b}_{t,n} \mid \mathcal{B}_\mathcal{C}\right)$ ▷ Coreset selection
    $\Theta \leftarrow \Theta - \eta \nabla f_\Theta(\mathcal{B}_t[\mathbf{u}^*] \cup \mathcal{B}_\mathcal{C})$ with Equation (9) ▷ Model update with selected instances
    $\mathcal{C}_t \leftarrow \mathcal{C}_t \cup \widehat{\mathcal{B}}_t$
  end for
  $\mathcal{C} \leftarrow \mathcal{C} \cup \textsc{Select}(\mathcal{C}_t, \text{size} = J/T)$ with Equation (8) ▷ Memorize coreset in the replay buffer
end for

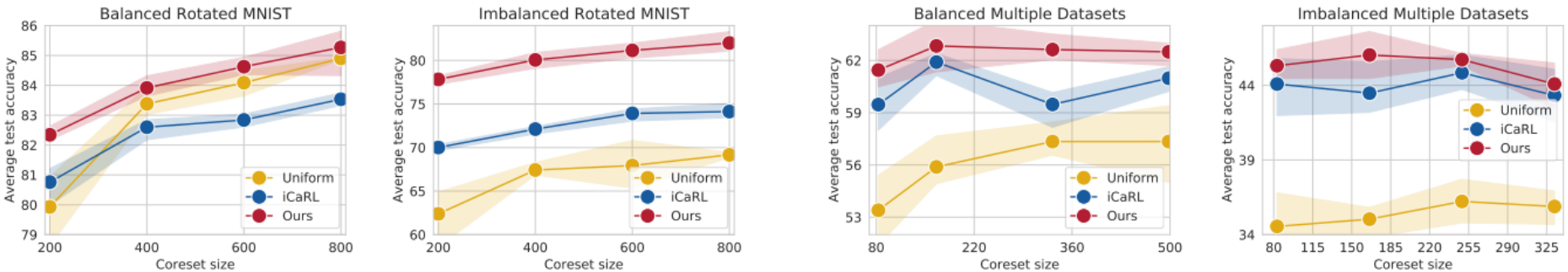
Experiments

The imbalanced setting contains a different number of training examples for each class in a task.



(a) Average test accuracy (b) First task test accuracy

Figure: (a) Average accuracy (b) First task accuracy for balanced/imbalanced Rotated MNIST during CL.

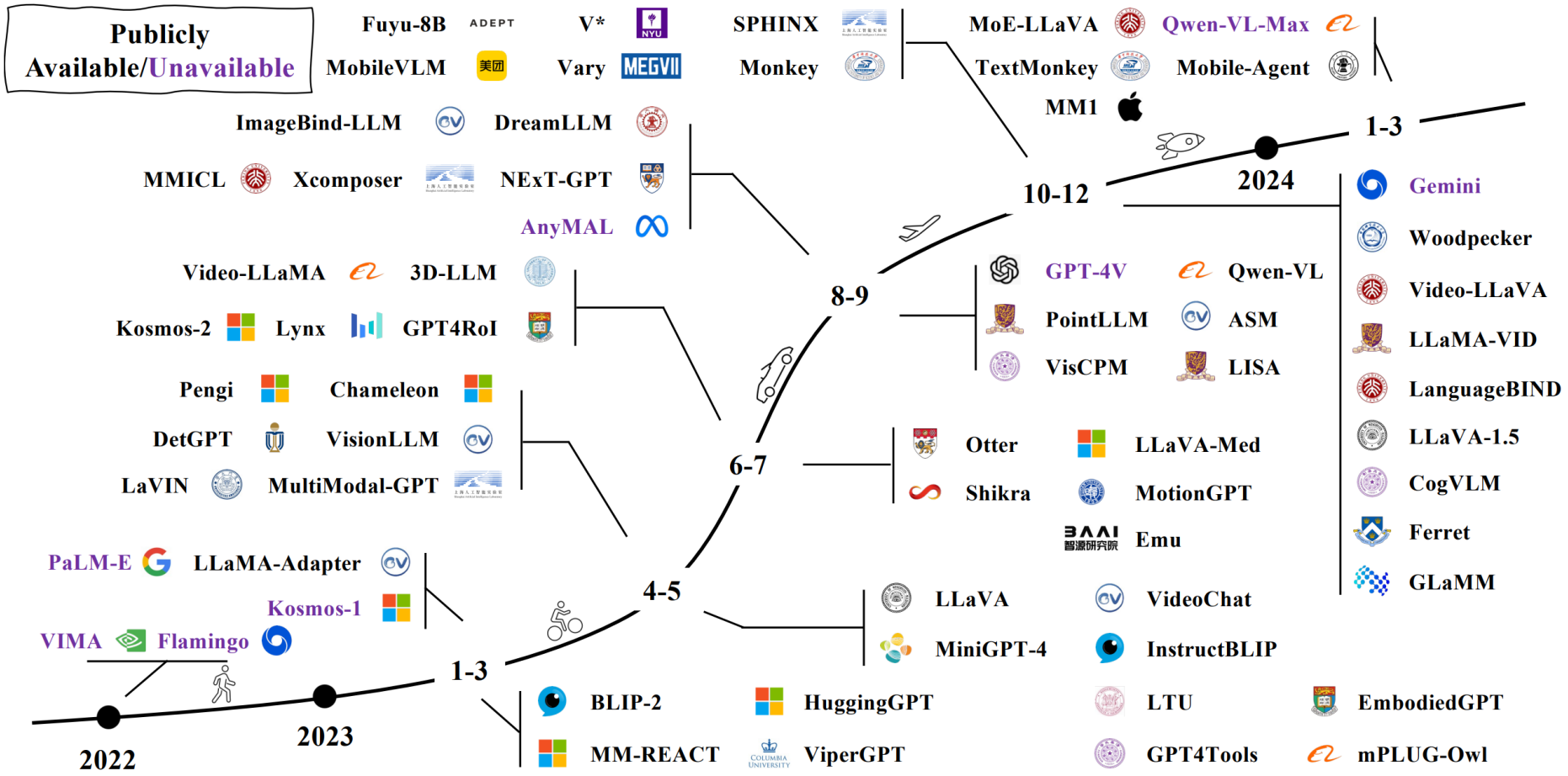


(a) Rotated MNIST (b) Multiple Datasets

Figure: Performance comparison on various coreset sizes for balanced/imbalanced continual learning

Introduction

Large Language Models (LLMs) are a type of advanced AI models designed to understand and generate human languages.



Efficient LLMs

Although LLMs are leading the next wave of AI revolution, their remarkable capabilities come at substantial resource demands

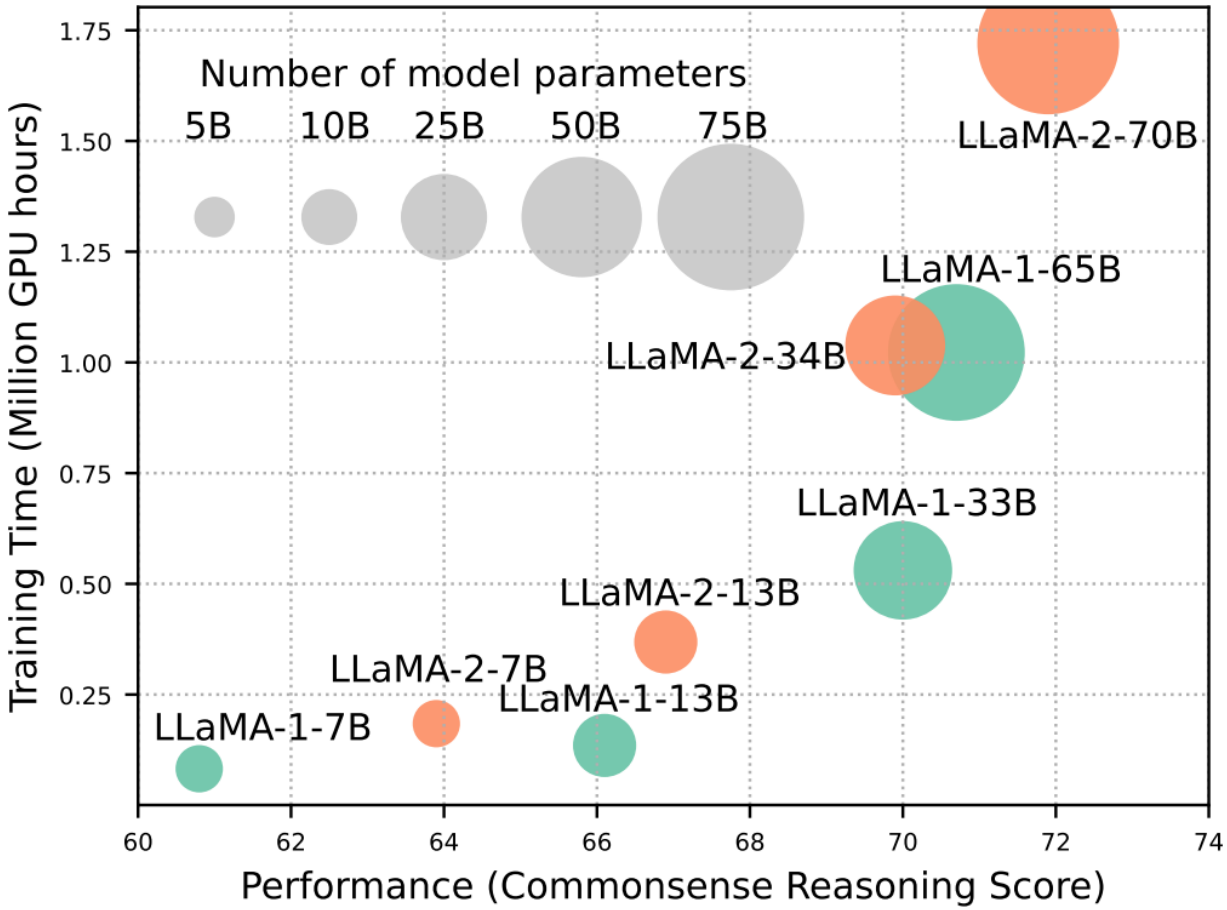


Figure: Illustration of model performance and model training time in GPU hours of LLaMA models at different scales.

Coresets in LLMs

Coreset selection is a fundamental technique for enhancing efficiency. As summarized in Figure below, in the context of LLMs, coreset selection techniques have been primarily used for enhancing the efficiency of pretraining and fine-tuning.

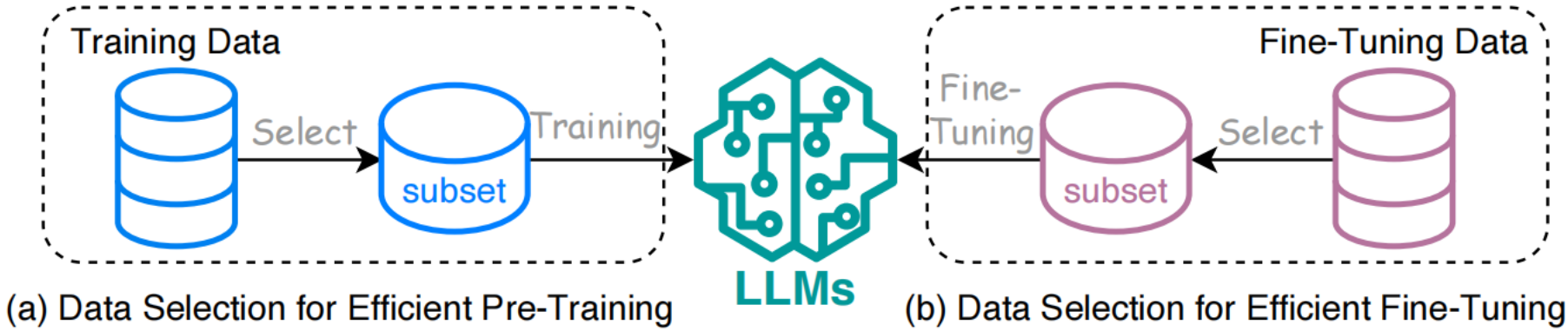


Figure: Illustrations of coreset selection techniques for LLMs.

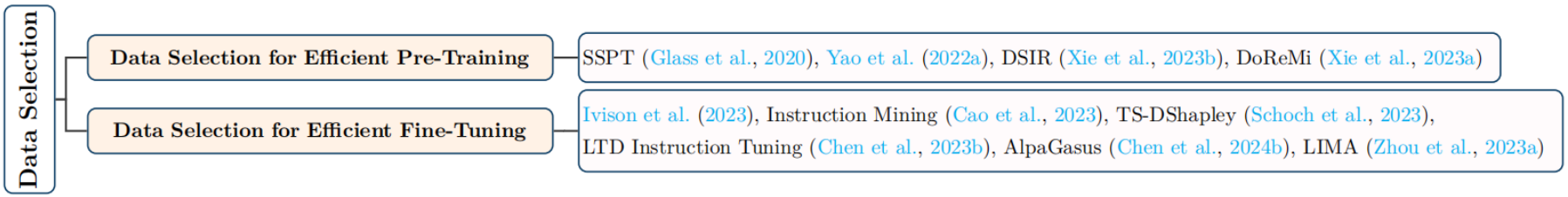


Figure: Summary of coreset selection techniques for LLMs.

CoLM

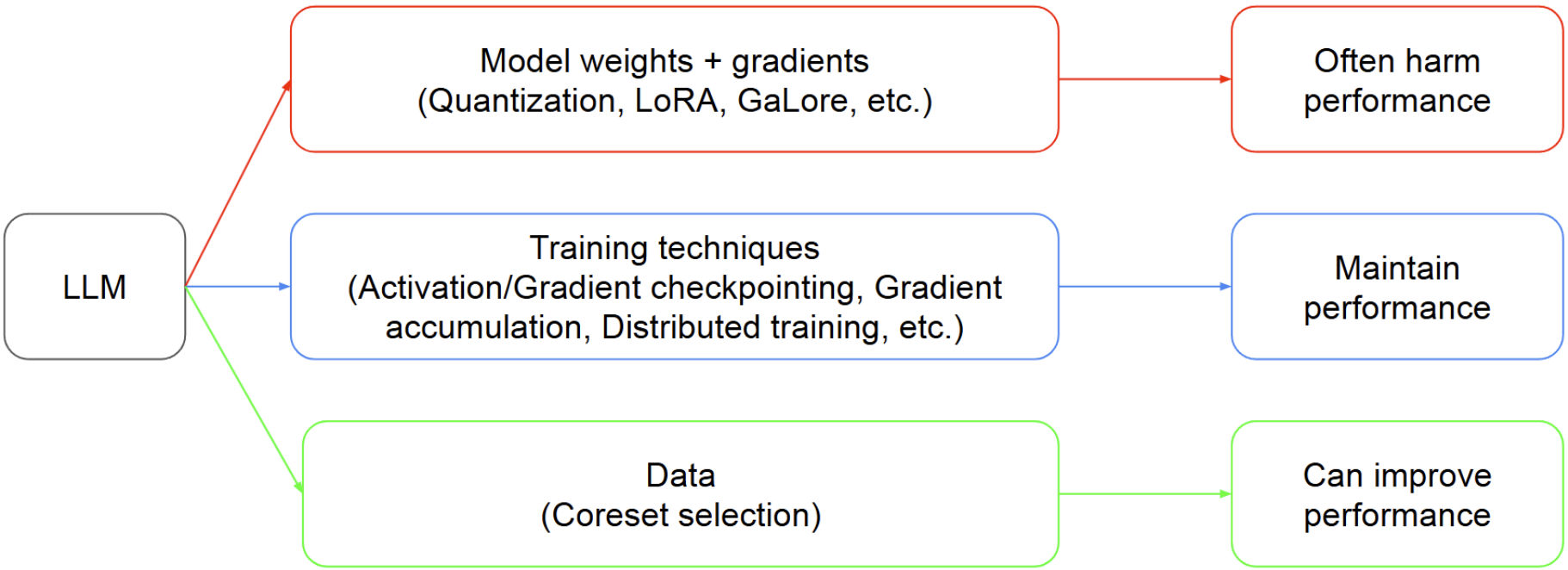
Published as a conference paper at ICLR 2025

MINI-BATCH CORESETS FOR MEMORY-EFFICIENT LANGUAGE MODEL TRAINING ON DATA MIXTURES

Dang Nguyen **Wenhan Yang** **Rathul Anand** **Yu Yang** **Baharan Mirzasoleiman**
{dangnth, hangeryang18, rathul, yuyang, baharan}@cs.ucla.edu
Computer Science Department, UCLA

CoLM

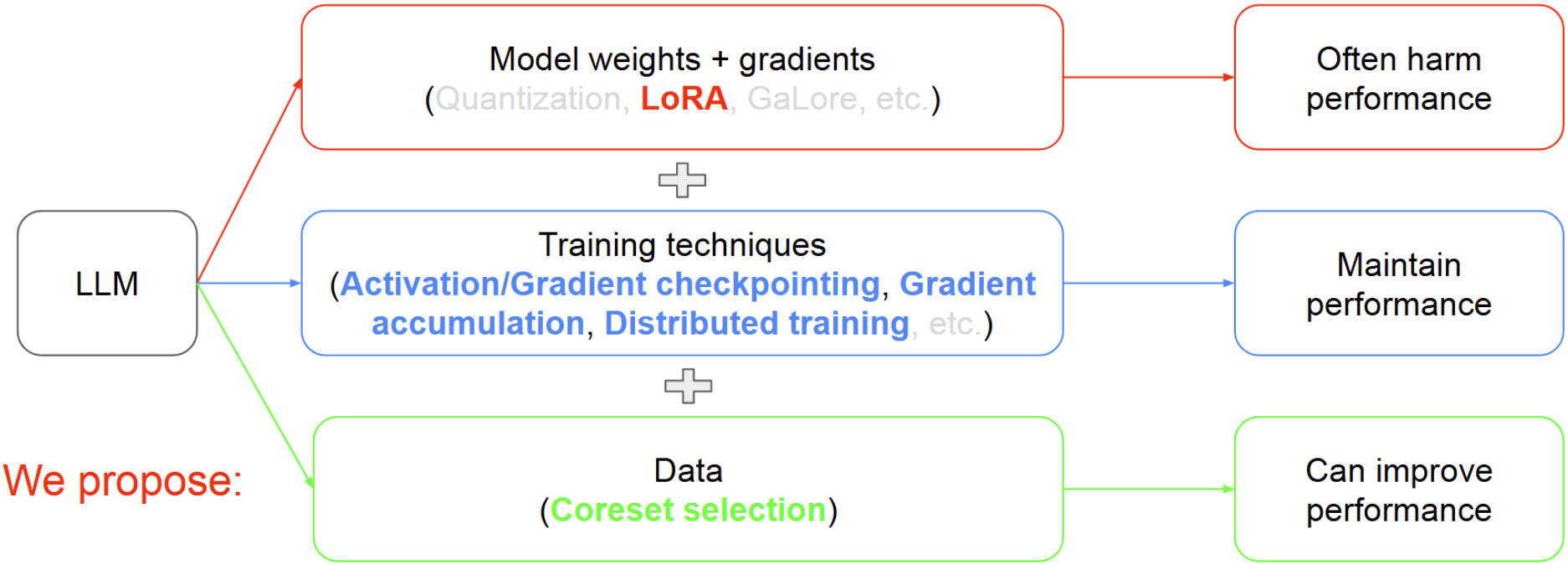
How to reduce GPU memory requirement in LLM?



CoLM

How to reduce GPU memory requirement in LLM?

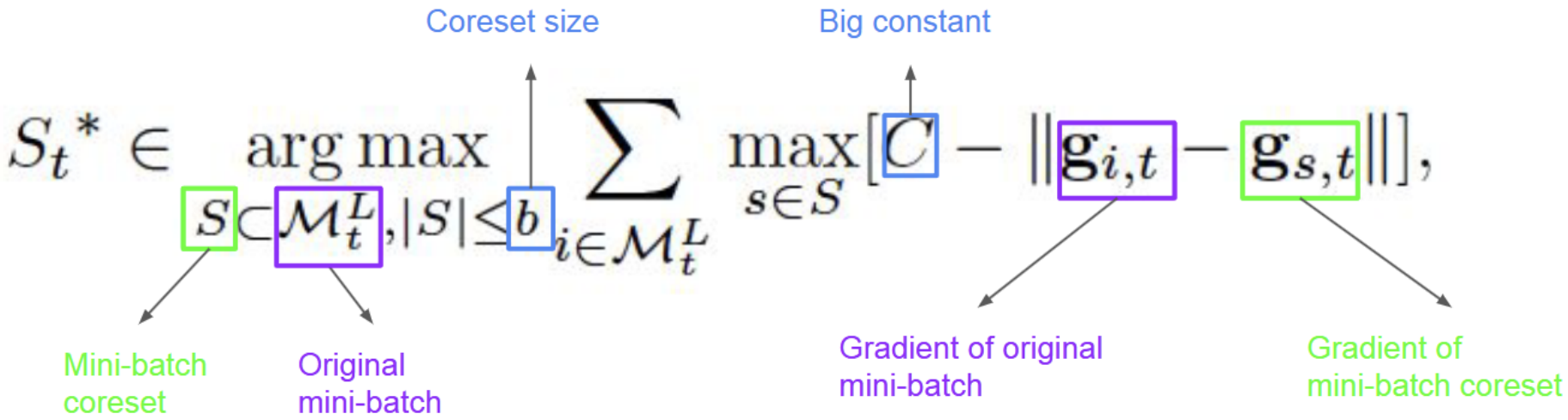
It's possible to stack three different approaches together.



CoLM

(Mini-batch) coreset selection

- Training with larger mini-batch has a small variance, thus convergences faster
- A mini-batch coreset is a subset that has the similar gradient to the original mini-batch. Thus, it can be found by solving the gradient matching problem



CoLM

Challenges of coreset selection for training LLMs

1. Highly Imbalanced Language Data
2. Adam optimizer
3. Very Large Gradient Dimensionality

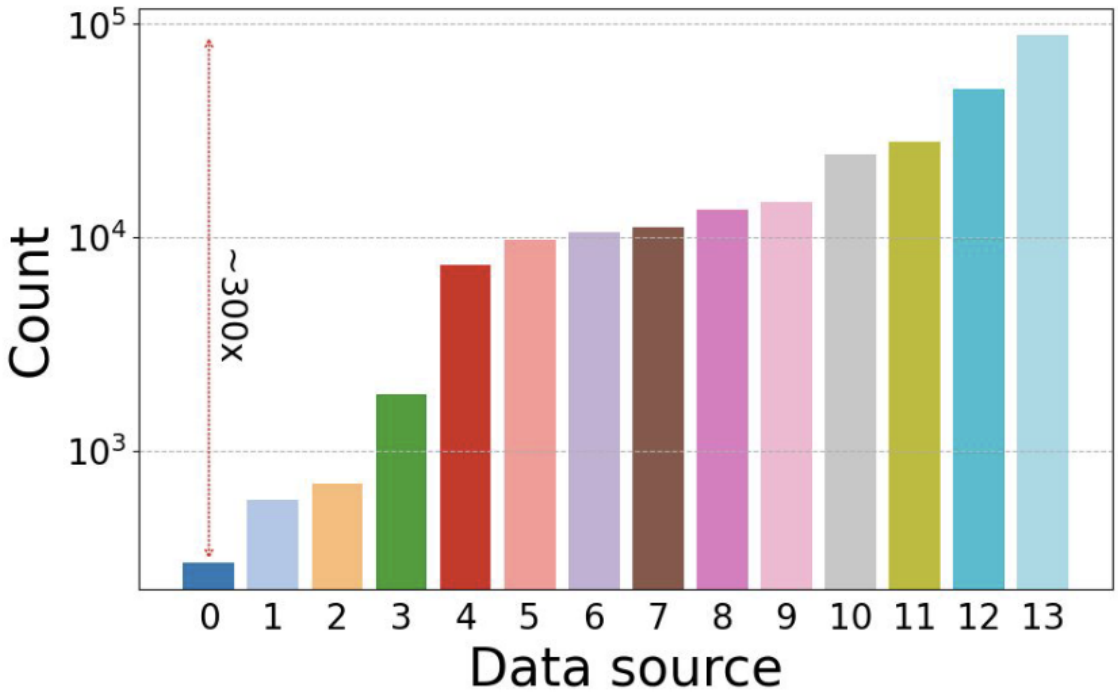


Figure: The number of samples from different data sources in MathInstruct.

CoLM

Coresets for Training LLMs (CoLM)

1. A balanced sampling strategy
2. Adam-like gradient normalization
3. Sparsified zeroth-order gradient estimation

Experimental results

CoLM yields the best of both worlds, increasing accuracy while reducing time & memory consumption.

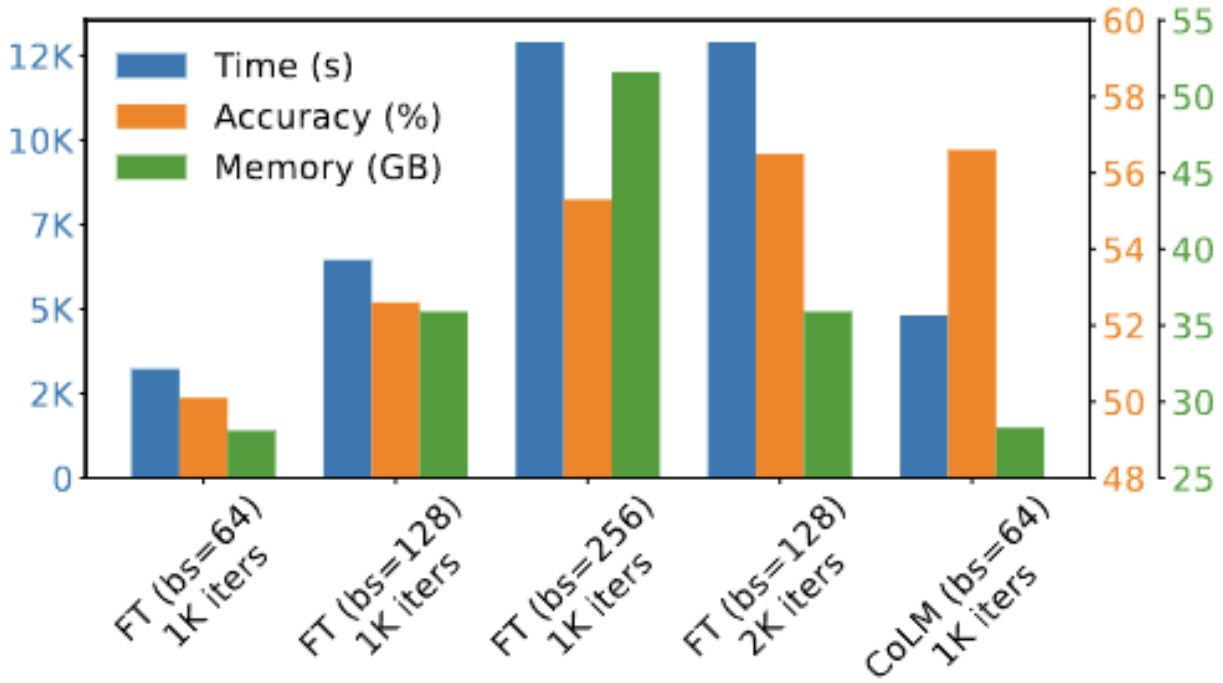
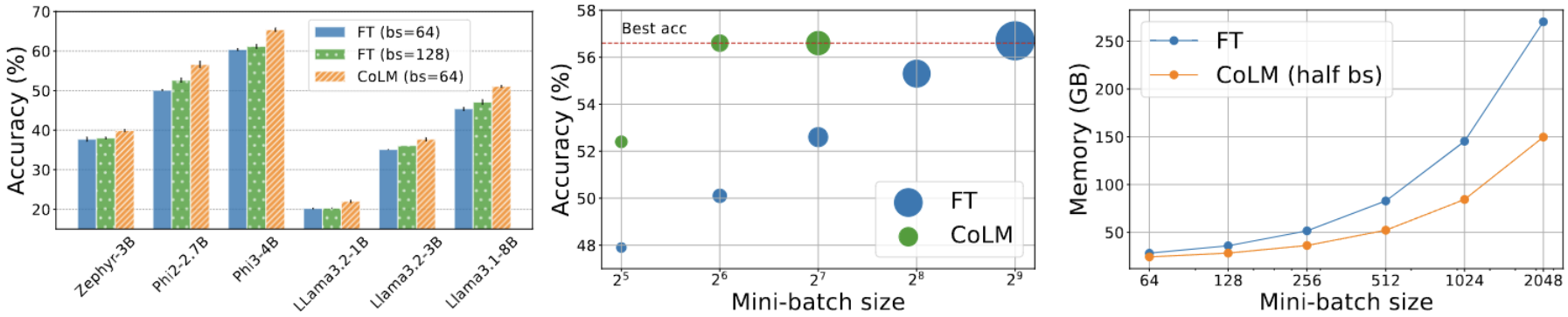


Figure: Time vs Accuracy vs Memory of fine-tuning Phi-2 with LoRA on MathInstruct.

Experimental results

CoLM improves the performance of different models and batch sizes.



(a) Different LLMs (b) Changing mini-batch size (c) Memory usage vs batch size

Figure: a) CoLM with bs = 64 (from 128) outperforms fine-tuning different models with bs = 64 and bs = 128 by a large margin; (b) CoLM improves the performance of training with different batch sizes. The size of each circle is proportional to the training time of the corresponding method. (c) CoLM reduces memory consumption, with reduction increasing as the batch size grows.

Challenges

1. Dynamic and Adaptive Coreset Selection
2. Automated and Multi-objective Coreset Optimization
3. Balancing Fidelity and Diversity
4. Scalability to Large-scale and Diverse Modalities
5. Ethical and Privacy Considerations