A Appendix

B Code

The code is submitted as a supplement along with the manuscript and will be released publicly upon acceptance.

C Notations

Symbol	Meaning
H	Input image height in pixels
W	Input image width in pixels
D	Network Depth
C	Number of convolutional channels (network width)
N	Number of models used during traning
M	Number of network features, proportional to C
$\|T\|$	Training set size
$\|B\|$	Training batch size
$\|S\|$	Support set/coreset size

Table 3: Notation for variables used throughout the paper. In some cases, C also refers to number of classes in the dataset (section 5), however it is clear from context.

D Implementation details

D.1 Preprocessing

For black/white datasets, i.e., MNIST and Fashion-MNIST, we use standard preprocessing, where we subtract the mean and divide by the standard deviation.

For color dataset SVHN, CIFAR-10, CIFAR-100, we use regularized Zero Component Analysis (ZCA) preprocessing with a regularization parameter of $\lambda = 0.1$ for all datasets. A description of this preprocessing method is available in the appendix of [40], or in [50]. For CIFAR-10 and CIFAR-100, we did not include the unit normalization step in regularized ZCA as we found it reduced performance by around 1-2%.

D.2 Architectures

For all architectures, we use the same ConvNet architecture used in [66, 64, 40] which consists of three layers of 3×3 convolutions, 2×2 average pool, and ReLU activations, followed by a fully connected layer. We do not use any normalization layers in any experiments unless otherwise stated.

For weight initialization, we use standard parameterization with Gaussian weight and bias initializations with variances $\sigma_w^2 = 2$ and $\sigma_b^2 = 0.1$, following [40]. Note that by default, PyTorch uses Kaiming uniform initializations, which means we had to write custom convolutional layers to have Gaussian initializations. While we did not collect data on this, we found this difference in initialization to be negligible - for all intents and purposes, the default initialization works just as well but corresponds to a slightly different NNGP process.

For RFAD training, we used neural networks with 256 convolutional channels per layer. Additionally, we removed the final fully-connected layer and used the representations after the final ReLU layer to calculate the NNGP kernel instead. This can be done by noting that for fully-connected layers $K^l = \sigma_w^2 K^{l-1} + \sigma_b^2$. This theoretically removes some variance associated with the final layer and saves on some memory. In practice, we found this did not affect performance.

D.3 Training

We used Adabelief optimizer [68] with a learning rate of $1e-3$ for all experiments and parameters, and $\varepsilon = 1e-16$.

Additionally, we found it useful to split up the representation of X_S into two pieces: one parameter $\hat{X}_S$ with the same shape as X_S: $\mathbb{R}^{|S| \times C \times H \times W}$ and another matrix $T \in \mathbb{R}^{(C \times H \times W) \times (C \times H \times W)}$. For example, for CIFAR-10, T would be a 3072×3072 matrix ($3072 = 3 \times 32 \times 32$). Then, we compute X_S as $X_S = \text{reshape}(T\text{flatten}(\hat{X}_S)$. Note this is like ZCA preprocessing where the transformation matrix is learned from T. In the code we refer to this as the `transform_mat`.

Note that this does not add any extra variables for the coreset, but in practice, we found that this trick resulted in slightly faster convergence, particularly at initialization. The T matrix is initialized at the identity and learned with a small learning rate (5e-5). While we do not have a theoretical justification for why this speeds up optimization, we believe it may allow the optimizer to learn dependencies between parameters, perhaps allowing it to behave more like a second-order optimizer.

The coreset is initialized with a class-balanced subset of the data. For the labels, we use one-hot vectors with $1/C$ subtracted so that the vector is zero-mean. E.g. for 10 samples per class the label would be $[0.9, -0.1, -0.1, ... - 0.1]$.

In experiments that use Platt scaling, we learn the logarithm of τ with a learning rate of 1e-2.

For each gradient step/training iteration, we use 5120 training examples. We compute features for these in 4 batches of 1280. We save memory in this step by calculating these features with the `torch.no_grad()` flag, as we do not backpropagate through these values. For the matrix inversion, we found it helpful to use double-precision since the process is sensitive to rounding errors.

Like with KIP, we used a adaptive kernel regularizer when computing the inverse in $(K_{SS} + \lambda I)^{-1}$. We parameterized this as $\lambda = \lambda_0 \bar{\text{tr}}(K_{SS})$, where $\bar{\text{tr}}$ is the average value along the diagonal. We used $\lambda_0 = 5e-3$.

We performed early stopping with the patience of 1000 iterations. Every 40 epochs, the loss on a validation set of size 1000 is measured. This validation set is a subset of the training set (we are training on the validation set); however, we used a fixed set of 16 random neural networks when calculating the validation loss.

For results in table 1, we ran the algorithm four separate times for each configuration.

D.4 Runtime experiments

The time taken was calculated by running RFAD on CIFAR-10 with images per class in $[1, 2, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]$, averaging over 200 training iterations with $N = 1, 2, 4, 8$. These experiments are run on a single RTX 3090. To make the results comparable to a V100, we add a conservative 40% extra time taken.

D.5 Finite Network Transfer

We use finite networks with 1024 convolutional channels with the same weight and bias-variance as we trained on, again with standard parameterization. We used SGD with a learning rate of either $1e-1, 1e-2, 1e-3, 1e-4$, momentum 0.9, weight decay of either $0, 1e-3$ and label scaling coefficients in $1, 2, 8, 16$. When weight decay was used, we used the modified weight decay $||\theta - \theta_0||_2^2$ given in [20], standard weight decay would not result in zero-output when centering is used. The best hyperparameters were determined by the best validation set accuracy (taking from the training set) on the first run of the algorithm. Unless otherwise stated, we use the centering trick. Batch sizes are up to 500, meaning that we performed full-batch gradient descent for all experiments except CIFAR-100 with 10 images per class.

The results reported in table 2 are the average of 12 training runs: 3 finite network training runs for each of the 4 repeat runs of RFAD.

Table 4: Total runtime and iteration for all RFAD distilled datasets. All experiments converge in under 10h on a single RTX 3090, with label learning usually taking longer.

		Fixed Labels			Learned Labels		
	Img/Cls	Number of iterations	Total elapsed time (h)	Average time per iteration (s)	Number of iterations	Total elapsed time (h)	Average time per iteration (s)
MNIST	1	6650 ± 859	1.6 ± 0.2	0.86	7390 ± 1907	1.7 ± 0.5	0.85
	10	11070 ± 3049	2.8 ± 0.8	0.90	12770 ± 2609	3.2 ± 0.6	0.90
	50	8330 ± 1225	2.7 ± 0.4	1.18	7420 ± 1047	2.4 ± 0.3	1.19
Fashion-MNIST	1	9580 ± 1740	2.3 ± 0.4	0.85	9390 ± 1924	2.2 ± 0.5	0.85
	10	14780 ± 2025	3.7 ± 0.5	0.89	13010 ± 2893	3.2 ± 0.7	0.89
	50	11190 ± 2056	3.6 ± 0.7	1.17	12230 ± 2206	3.9 ± 0.7	1.15
SVHN	1	7540 ± 1521	3.4 ± 0.7	1.60	5700 ± 1267	2.5 ± 0.6	1.59
	10	9060 ± 2155	4.2 ± 1.0	1.67	7330 ± 1954	3.4 ± 0.9	1.66
	50	8270 ± 2717	4.8 ± 1.5	2.10	10370 ± 758	6.0 ± 0.5	2.08
CIFAR-10	1	4610 ± 778	2.0 ± 0.3	1.56	4300 ± 1457	1.9 ± 0.6	1.60
	10	8310 ± 2096	3.8 ± 1.0	1.64	9210 ± 923	4.3 ± 0.4	1.66
	50	8370 ± 2335	4.9 ± 1.3	2.13	14140 ± 4901	8.4 ± 3.0	2.13
CIFAR-100	1	9820 ± 2067	4.7 ± 1.0	1.71	13650 ± 4161	6.1 ± 2.3	1.62
	10	8410 ± 2069	6.3 ± 1.5	2.72	13610 ± 1463	9.6 ± 0.7	2.54

D.6 Privacy experiments

For CIFAR-10 corruption experiments, we use the same training protocol as in appendix D.3. Rather than initializing the coreset with real images, we initialize with white noise in image space. The corruption constraint is applied in *pixel* space rather than the ZCA space. Note that the corruption mask (whether a pixel can be optimized) is done per-pixel and per-channel, meaning that a pixel can have its red and green channels fixed but blue channel free.

We downsized the CelebA dataset to 64×64 images, applying standard preprocessing for this experiment. We used a training batch size of 1280 for CelebA. The full dataset achieves an accuracy of 97.6%.

For the privacy experiments, we ran only a single run for each corruption ratio. To calculate the means and standard deviations, we took the best iteration, based on the early stopping condition, and iterations at 400, 200 iterations before, and 200 and 400 after.

E Time taken additional results

As discussed in section 4.1 and appendix D.4, we added an extra 40% to iteration times to make or times on a RTX 3090 comparable to a Tesla V100. In this section, we report the original times. Additionally, we report the total number of training iterations and total runtimes for all our distillation results. Note that because of the early stopping condition, the epochs used during evaluation are 1000 iterations before the iteration counts we report here.

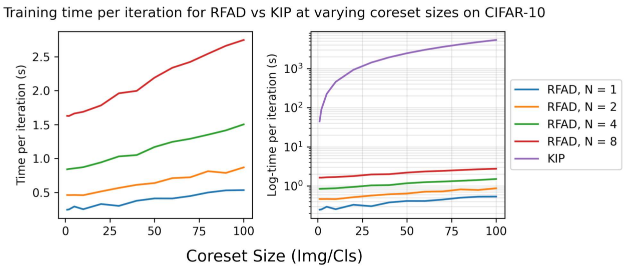


Figure 9: Time per training iteration for CIFAR-10 with varying number of models used during and and support sizes run on an RTX 3090 compared to KIP

F Centering and label scaling for finite networks ablations

table 5 shows the full table finite network transfer results. We see that label scaling and centering provide performance benefits, particularly for small support sets. We do not have a theoretical explanation for why label scaling improves performance. [11] suggests that label scaling for values $\alpha < 1$ should result in the network staying in the lazy regime, but in our case, we found values of $\alpha > 1$ to improve performance.

	Img/Cls	Fixed labels no label scale	Fixed labels no centering	Fixed labels	Learned labels
MNIST	1	84.1 ± 5.5	90.1 ± 3.3	92.2 ± 2.1	94.4 ± 1.5
	10	96.7 ± 0.3	98.3 ± 0.2	98.4 ± 0.2	98.5 ± 0.1
	50	98.5 ± 0.2	98.8 ± 0.1	98.8 ± 0.1	98.8 ± 0.1
Fashion-MNIST	1	51.0 ± 19.2	69.8 ± 3.4	76.7 ± 1.7	78.6 ± 1.3
	10	84.5 ± 0.9	85.0 ± 0.7	87.0 ± 0.5	85.9 ± 0.7
	50	87.7 ± 0.4	87.4 ± 0.4	88.8 ± 0.4	88.5 ± 0.4
SVHN	1	32.6 ± 3.0	40.2 ± 2.9	43.1 ± 2.4	52.2 ± 2.2
	10	65.7 ± 1.0	72.9 ± 0.8	73.6 ± 1.0	74.9 ± 0.4
	50	78.0 ± 0.3	78.5 ± 0.3	80.1 ± 0.4	80.9 ± 0.3
CIFAR-10	1	48.7 ± 1.6	48.0 ± 1.7	53.2 ± 1.2	53.6 ± 0.9
	10	63.4 ± 0.6	56.9 ± 1.0	66.1 ± 0.5	66.3 ± 0.5
	50	69.5 ± 0.4	68.0 ± 0.6	71.1 ± 0.4	70.3 ± 0.5
CIFAR-100	1	19.6 ± 0.6	21.2 ± 0.3	24.2 ± 0.4	26.3 ± 1.1
	10	30.5 ± 0.3	18.5 ± 0.6	30.3 ± 0.3	33.0 ± 0.3

Table 5: Finite network transfer performance of KIP distilled images. We additional report performance if either label scaling or centering is not used. Label scaling and centering provide benefits particularly for smaller support set sizes.

G Effect of InstanceNorm

In section 7 we discussed the observation that if we attempt to use instancenorm for finite network training for KIP distilled images, we do not see good performance. Conversely, if we use random networks with instancenorm in RFAD, these distilled datasets do not perform well on networks without instancenorm, either in the KRR case or finite network case. table 6 shows the exact results. For NNGP KRR with instancenorm, we used the empirical NNGP kernel, with 32 networks with 1024 channels, as there is no exact implementation of the instancenorm NNGP in the neural-tangents PyTorch library [42].

H Interpretability additional examples

I Empirical NNGP at Inference additional results

This section contains additional plots showing the effectiveness of using the Empirical NTK at inference for all of our RFAD distilled datasets. In all cases, we are able to achieve close to the performance of the exact NNGP kernel for convolutional architectures with $C \geq 128$. We additionally show an experiment where we achieve 70% accuracy on CIFAR 10, using our 10 img/cls fixed label distilled coreset using the empirical NNGP kernel from random neural networks with *one* convolutional channel in fig. 12

		Without InstanceNorm		With InstanceNorm	
	Img/Cls	NNGP KRR	Finite Network	NNGP KRR*	Finite Network
Trained without InstanceNorm	1	61.1 ± 0.7	53.2 ± 1.2	35.3 ± 0.9	37.4 ± 1.1
	10	73.1 ± 0.1	66.1 ± 0.5	45.9 ± 1.8	45.1 ± 1.1
	50	76.1 ± 0.3	71.1 ± 0.4	59.1 ± 0.4	50.3 ± 0.8
Trained with InstanceNorm	1	18.1 ± 3.7	40.6 ± 3.7	57.8 ± 0.7	52.8 ± 0.7
	10	25.6 ± 5.3	36.3 ± 1.5	71.1 ± 0.2	63.5 ± 0.5
	50	52.5 ± 0.5	55.0 ± 0.6	74.4 ± 0.2	62.2 ± 0.4

Table 6: Accuracy of RFAD distilled datasets run with or without networks with instancenorm during training evaluated on NNGP KRR and finite networks with SGD with or without instancenorm. We see that transferring from instancenorm to no instancenorm or vice versa incurs a large performance penalty. * indicates thats the emperical NNGP kernel was used, as there is no exact implementation of instancenorm in the neural-tangents library [42]

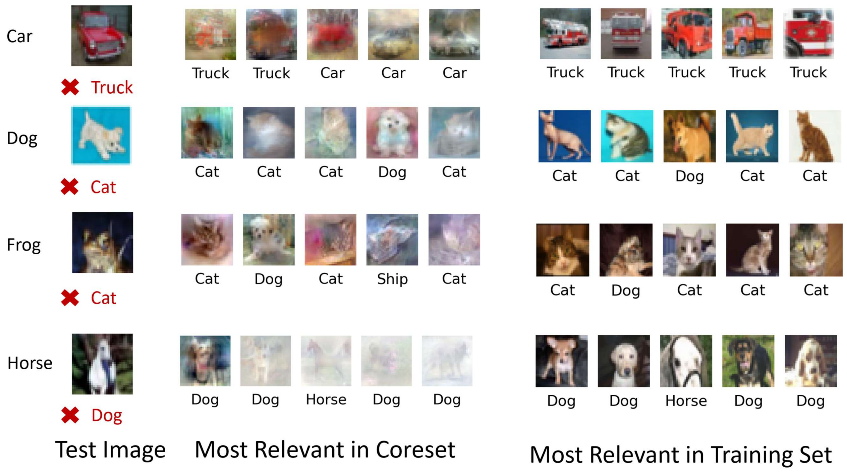


Figure 10: Incorrectly predicted test images and the most relevant images in the coreset and training set for CIFAR-10, 50 images/cls

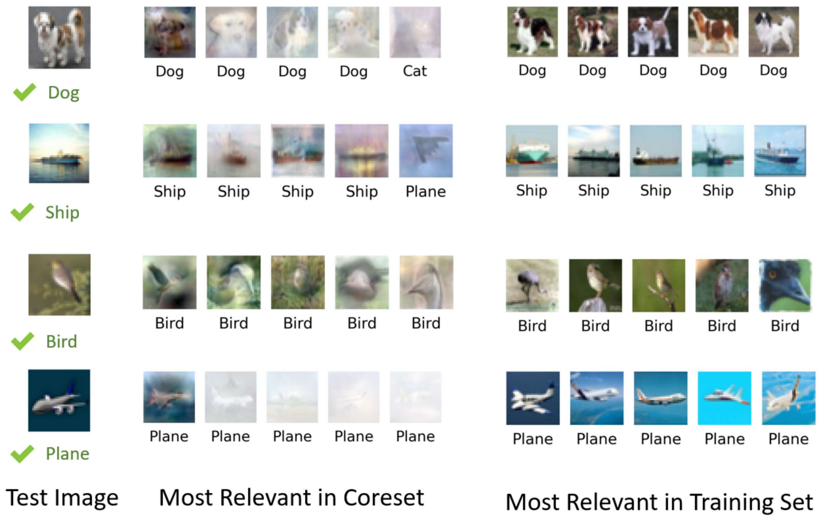


Figure 11: Correctly predicted test images and the most relevant images in the coreset and training set for CIFAR-10, 50 images/cls

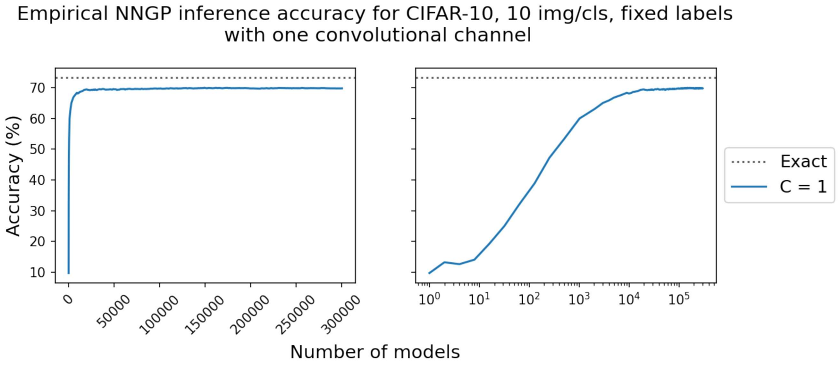


Figure 12: Empirical NNGP performance at inference for CIFAR-10, 10 images per class, fixed labels, using convolutional networks with one convolutional channels. We can achieve reasonable performance, 70%, albeit requiring over 10^5 random models.

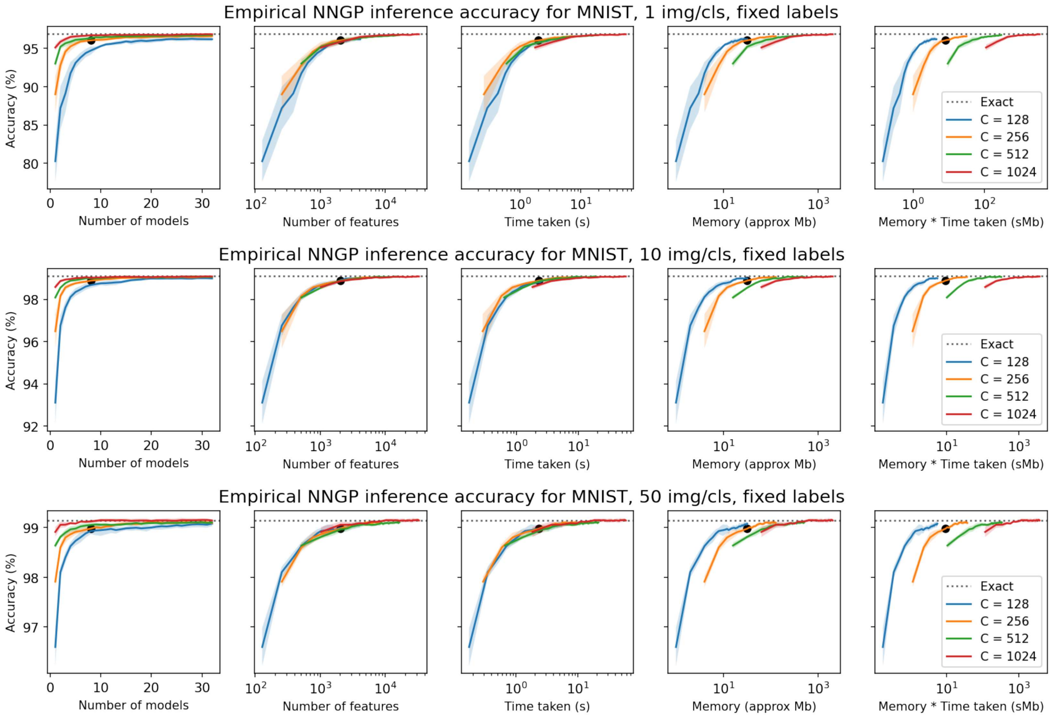


Figure 13: Empirical NNGP inference accuracy for MNIST with fixed labels

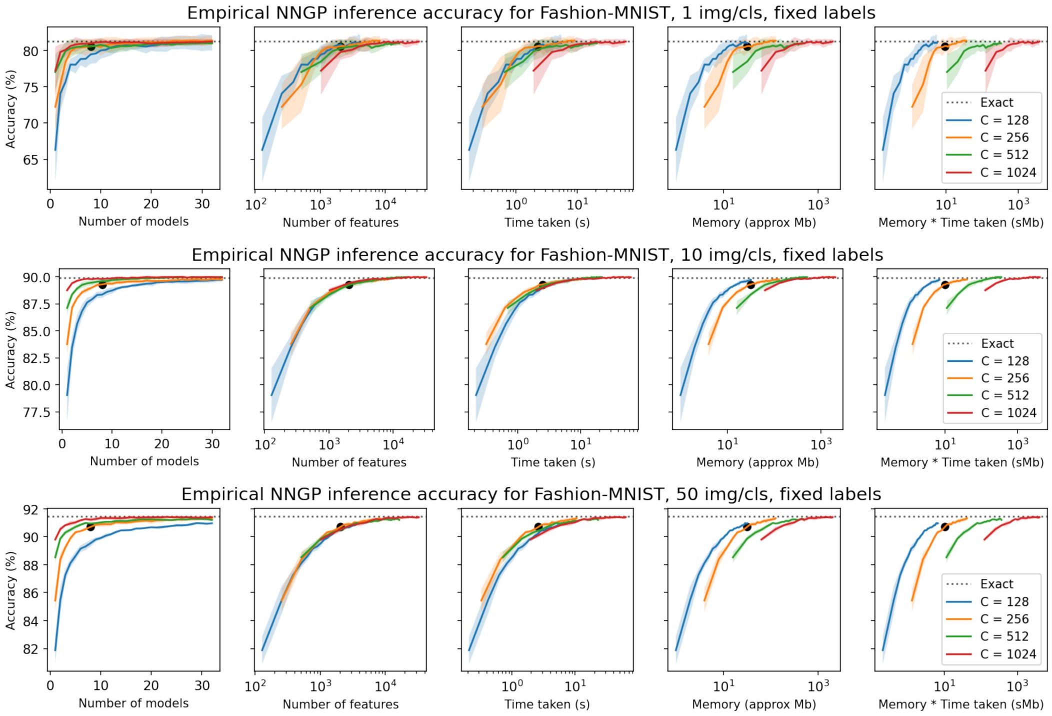


Figure 14: Empirical NNGP inference accuracy for Fashion-MNIST with fixed labels

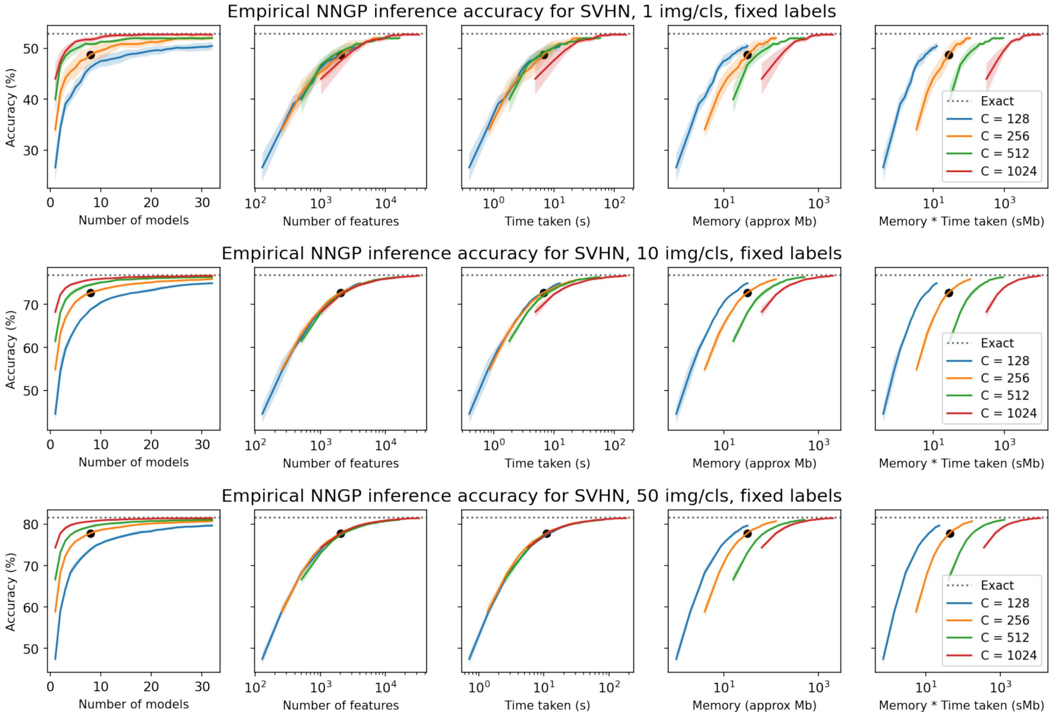


Figure 15: Empirical NNGP inference accuracy for SVHN with fixed labels

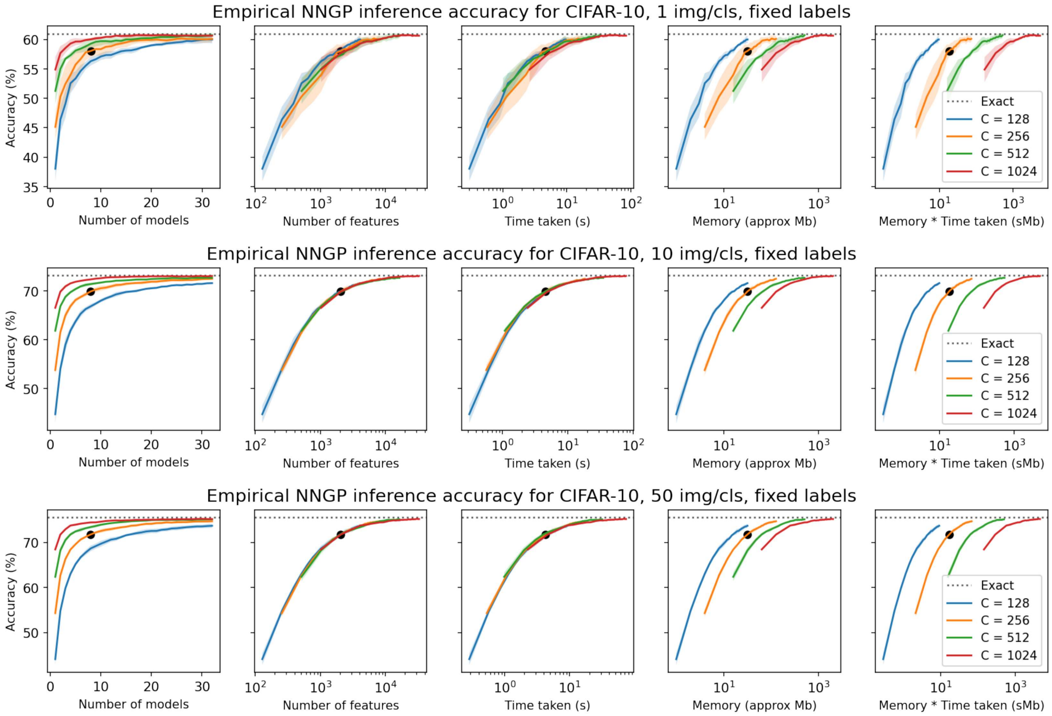
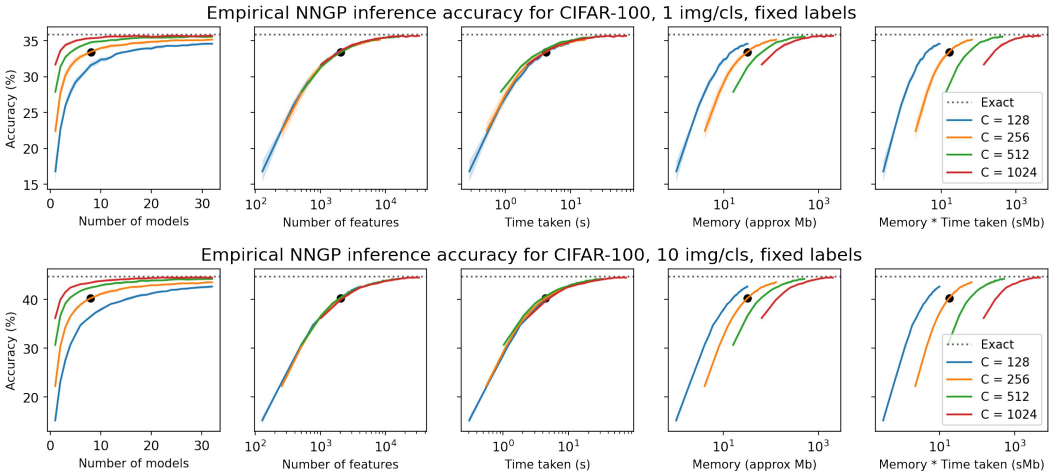
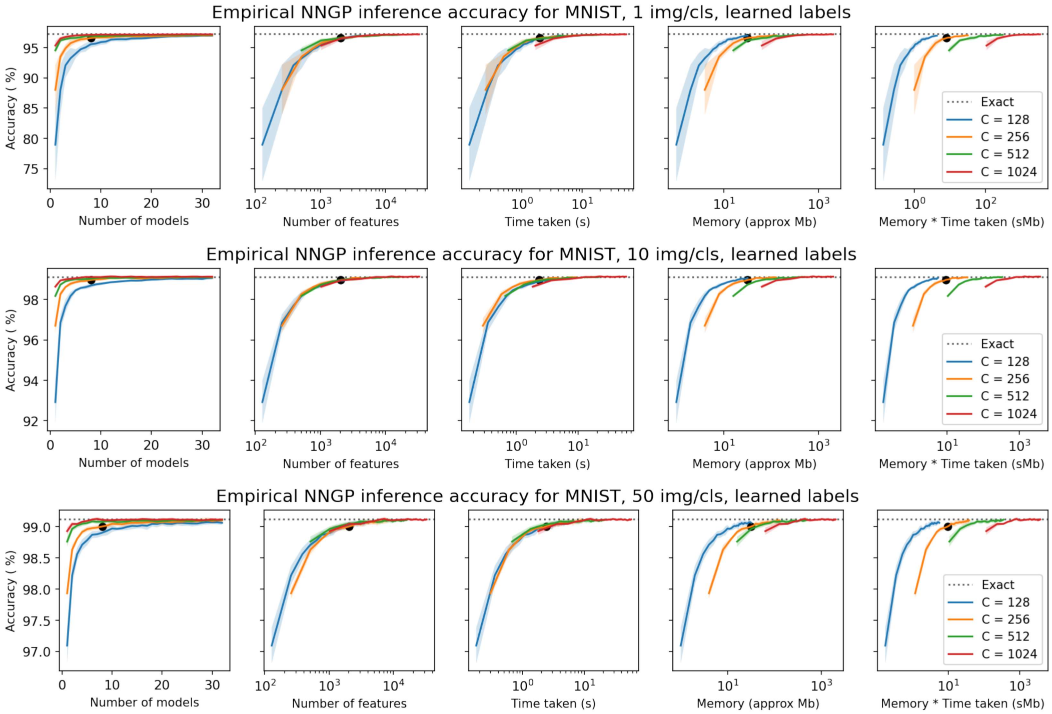


Figure 16: Empirical NNGP inference accuracy for CIFAR-10 with fixed labels


Empirical NNGP inference accuracy for CIFAR-100, 1 img/cls, fixed labels

Empirical NNGP inference accuracy for CIFAR-100, 10 img/cls, fixed labels

Figure 17: Emperical NNGP inference accuracy for CIFAR-100 with fixed labels


Empirical NNGP inference accuracy for MNIST, 1 img/cls, learned labels

Empirical NNGP inference accuracy for MNIST, 10 img/cls, learned labels

Empirical NNGP inference accuracy for MNIST, 50 img/cls, learned labels

Figure 18: Empirical NNGP inference accuracy for MNIST with learned labels

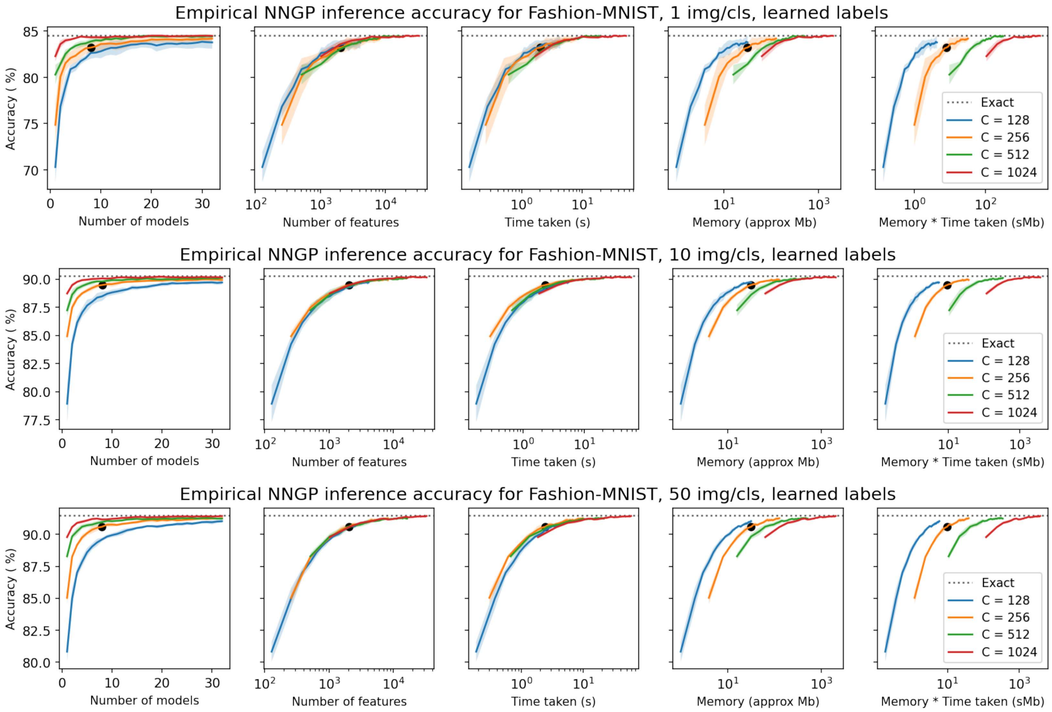


Figure 19: Empirical NNGP inference accuracy for Fashion-MNIST with learned labels

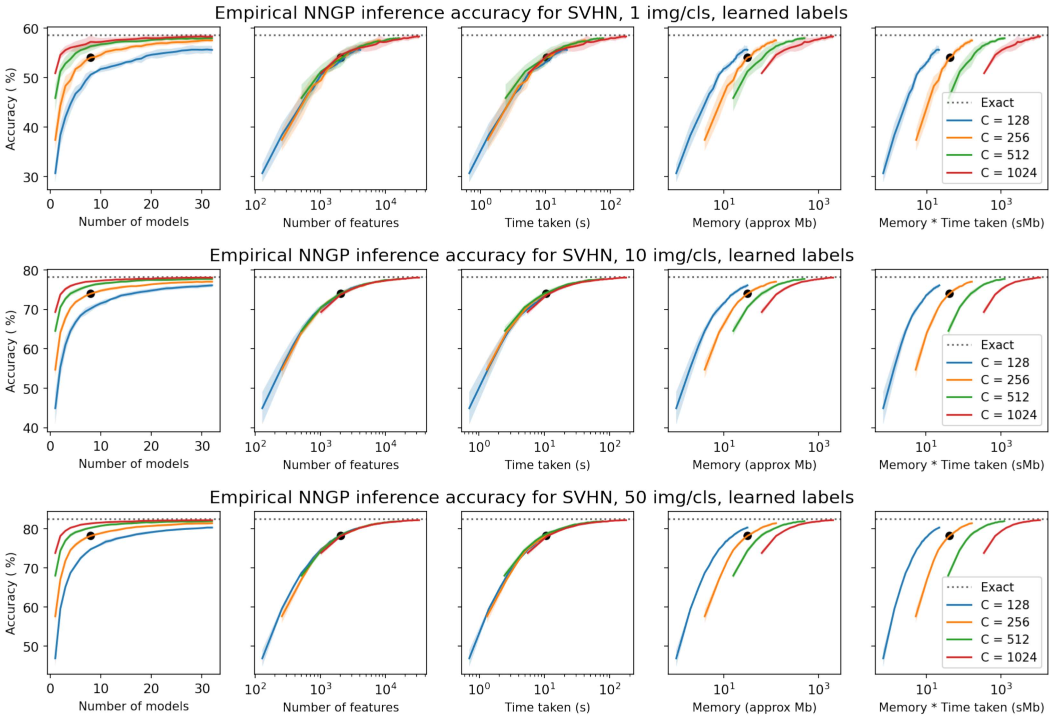
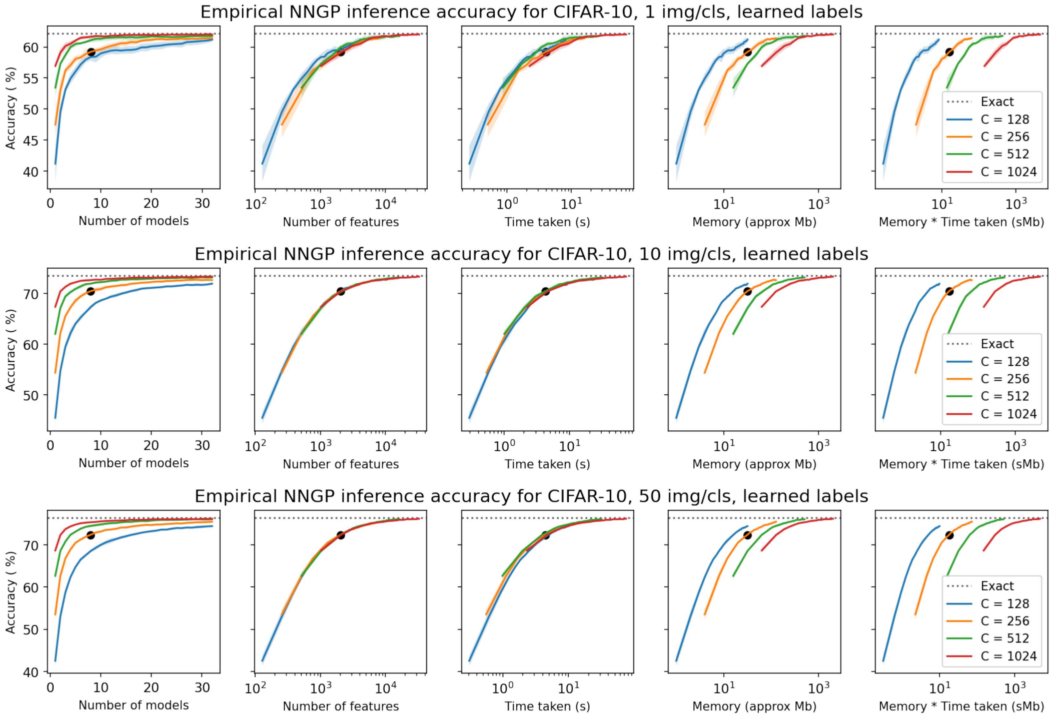


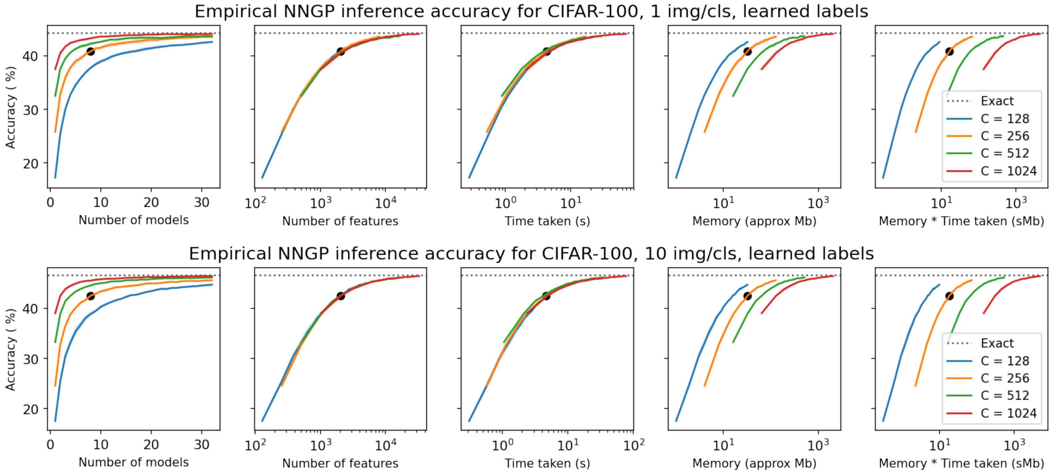
Figure 20: Empirical NNGP inference accuracy for SVHN with learned labels


Empirical NNGP inference accuracy for CIFAR-10, 1 img/cls, learned labels

Empirical NNGP inference accuracy for CIFAR-10, 10 img/cls, learned labels

Empirical NNGP inference accuracy for CIFAR-10, 50 img/cls, learned labels

Figure 21: Empirical NNGP inference accuracy for CIFAR-10 with learned labels


Empirical NNGP inference accuracy for CIFAR-100, 1 img/cls, learned labels

Empirical NNGP inference accuracy for CIFAR-100, 10 img/cls, learned labels

Figure 22: Emperical NNGP inference accuracy for CIFAR-100 with learned labels