A FLOW-RBC

Even given the rapidly growing enthusiasm around single-cell data, Flow-RBC constitutes a dataset unique for its kind as it consists of more than 100,000 measurements taken on different patients paired with a clinical label. Even established projects like the Human Cell Atlas (Regev et al., 2017) or Flow Repository[3] do not include single-cell datasets of this size. For instance, to our knowledge, the second largest open-source dataset of single-cell blood samples contains data from 2,000 individuals and does not include external clinical outcomes for all patients to be used as a target.

Flow-RBC consists of 98,240 train and 23,104 test sets. Each input set is a red blood cell (RBC) distribution of 1,000 cells. Each cell consists of a volume and hemoglobin content measurement (see Figure 4 for a visual representation). The regression task consists of predicting the corresponding hematocrit level measured on the same blood sample. Blood consists of different components: red blood cells, white blood cells, platelets and plasma. The hematocrit level measures the percentage of volume taken up by red blood cells in a blood sample.

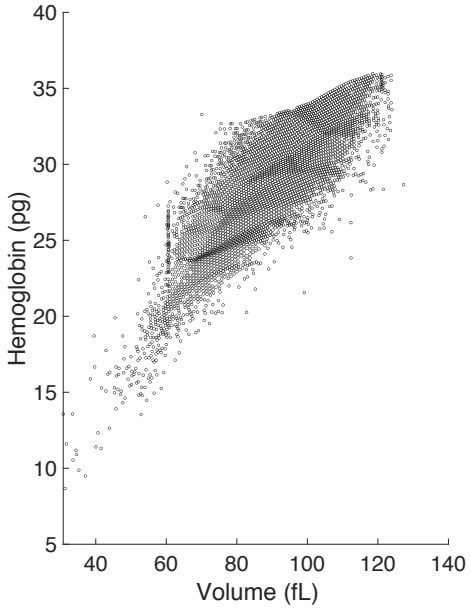


Figure 4: Example of RBC distribution given in input for the prediction of hematocrit level.

Since we only have information about the volume and hemoglobin of individual RBCs and no information about other blood cells, this task aims to answer an interesting clinical question: is there information present in individual RBC volume and hemoglobin measurements about the overall volume of RBCs in the blood? As this question has not been definitively answered in the literature, there is no known expected performance achievable; instead, increases in performance are an exciting scientific signal, suggesting a stronger relationship between single cell RBC and aggregate population properties of the human blood than previously known.

The existing scientific literature notes, in the presence of diseases like anemia, there exists a negative correlation between hematocrit and the red cell distribution width (RDW), also known as the coefficient of variation of the volume i.e. SD(Volume) / Mean(Volume) (McPherson et al., 2021, Chapter 9). We report as a baseline the MSE performance of a linear regression model with RDW as covariate. We used a linear model to represent the current state of medical knowledge on this topic. Additionally, we build a regression model on hand-crafted statistics of the distribution (up to the fourth moment on both marginal distributions as well as .1, .25, .5, .75, .9 quantiles). This model improves over simple prediction with RDW, further sustaining the hypothesis that more information

[3]https://flowrepository.org

lies in the single-cell measurements of RBCs. ST++ further improves performance, resulting in an MSE reduction of 28% over the RDW model. See Table 5 for results.

Procedure to Obtain RBC distribution measurements All Flow-RBC data is collected retrospectively at *hidden-for-anonymity* under an existing IRB-approved research protocol and is available at *hidden-for-anonymity*. Each RBC distribution consists of volume and hemoglobin mass measurements collected using the Advia 2120 (Harris et al., 2005), a flow cytometry based system that measures thousands of cells. The volume and hemoglobin information are retrieved through Mie (or Lorenz-Mie) theory equations for the analysis of light scattering from a homogeneous spherical particle (Tycko et al., 1985). An example of one input distribution is provided in Figure 4. The Advia machine returns an average of 55,000 cells. For this dataset, we downsampled each distribution to 1,000 cells, a number high enough to maintain reasonable-variance sample estimators of the "population" values (i.e. from 55,000 cells) while imposing reasonable memory requirements on consumer gpus. Each distribution is normalized and re-scaled by the global train mean and standard deviation.

Table 5: Baseline regression performances for the prediction of hematocrit from RBC distributions. Our proposed Set Transformer++ currently has the best performance on this task.

	MSE
RDW	25.85
Moments	22.31
Set Transformer++	18.69

B Normalization Proofs

Proposition 1. *Per feature transformation is the only setting which maintains permutation equivariance and prediction behavior which is agnostic to batch position.*

Proof. For transformation tensors in $\mathbb{R}^{B\times S\times D}$, the parameters can be distinct over the batch ($B \in M$), over the samples $S \in M$, over the features ($D \in M$), or any combination of the three.

Having distinct parameters over the samples breaks permutation equivariance, making $S \in M$ an untenable option. Let $f : \mathbb{R}^{S\times D} \to \mathbb{R}^{S\times D}$ be the transformation function, and $\overrightarrow{\gamma}_{\{S\}}, \overrightarrow{\beta}_{\{S\}}$ represent tensors in $\mathbb{R}^{B\times S\times D}$ where the values along dimension S can be unique, while the values along B, D are repeated. We denote an indexing into the batch dimension as $\overrightarrow{\gamma}_{\{S\},b}, \overrightarrow{\beta}_{\{S\},b}$. Then, f breaks permutation equivariance:

$$f(\pi\mathbf{a}) = \pi\mathbf{a} \odot \overrightarrow{\gamma}_{\{S\},b} + \overrightarrow{\beta}_{\{S\},b} \tag{4}$$
$$\neq \pi(\mathbf{a} \odot \overrightarrow{\gamma}_{\{S\},b} + \overrightarrow{\beta}_{\{S\},b}) \tag{5}$$
$$= \pi f(\mathbf{a}). \tag{6}$$

Having distinct parameters over the batch means that the position of a set in the batch changes its ordering, making $B \in M$ an untenable option. Let $\overrightarrow{\gamma}_{\{B\}}, \overrightarrow{\beta}_{\{B\}}$ represent tensors which can differ over the batch, e.g. $\overrightarrow{\gamma}_{\{B\},b} \neq \overrightarrow{\gamma}_{\{B\},b'}, b \neq b'$. Then, the prediction function f_b for batch index b will yield a different output than the prediction function $f_{b'}$ for batch index b':

$$f_b(\mathbf{a}) = \mathbf{a} \odot \overrightarrow{\gamma}_{\{B\},b} + \overrightarrow{\beta}_{\{B\},b} \tag{7}$$
$$\neq \mathbf{a} \odot \overrightarrow{\gamma}_{\{B\},b'} + \overrightarrow{\beta}_{\{B\},b'} \tag{8}$$
$$= f_{b'}(\mathbf{a}). \tag{9}$$

As neither S nor B can be in M, the remaining options are $M = \{D\}$ or $M = \{\}$, i.e. $\overrightarrow{\gamma}, \overrightarrow{\beta}$ each repeat a single value across the tensor. Note that $M = \{\}$ is strictly contained in $M = \{D\}$: if the

per feature parameters are set to be equal in the $M = \{D\}$ setting, the result is equivalent to $M = \{\}$. Therefore, $M = \{D\}$ sufficiently describes the only suitable setting of parameters for transformation.

□

Proposition 4. *Applying standardization on each sample separately over the batch (per sample and per sample per feature standardization) breaks permutation equivariance.*

Proof. We show that, given a batch size greater than 1, permuting one set in a batch changes the per sample statistics over the batch, breaking permutation equivariance.

For $L = \{S\}$:

$$f(\pi \mathbf{a}) = \frac{\pi \mathbf{a} - \overrightarrow{\mu}_{\{S\}}}{\overrightarrow{\sigma}_{\{S\}}} \tag{10}$$

$$\neq \frac{\pi \mathbf{a} - \pi \overrightarrow{\mu}_{\{S\}}}{\pi \overrightarrow{\sigma}_{\{S\}}} \tag{11}$$

$$= \pi f(\mathbf{a}). \tag{12}$$

For $L = \{S, D\}$:

$$f(\pi \mathbf{a}) = \frac{\pi \mathbf{a} - \overrightarrow{\mu}_{\{S,D\}}}{\overrightarrow{\sigma}_{\{S,D\}}} \tag{13}$$

$$\neq \frac{\pi \mathbf{a} - \pi \overrightarrow{\mu}_{\{S,D\}}}{\pi \overrightarrow{\sigma}_{\{S,D\}}} \tag{14}$$

$$= \pi f(\mathbf{a}). \tag{15}$$

These settings are equivariant to a consistent permutation across all sets in the batch, but the reliance on coordination between sets for permutation equivariance makes these settings unfavorable. To avoid dependence between inputs in a batch during test time, normalization layers such as batch norm use fixed mean and variance statistics based on a running average during training time. However, while this setting reduces undesirable dependence between inputs in a batch, it also is one where permutation equivariance is broken; the standardization statistics used at one position in a set may not match that of another, and permuting samples means that the individual samples are now being altered with different statistics. □

Proposition 2. *Applying layer norm in its most common placement (after linear projection, before non-linearity)*[4] *removes mean and variance information from each sample.*

Proof. Layer norm is typically placed after a linear projection and before the non-linear activation (Ba et al., 2016). This placement choice aligns with other normalizations (Ioffe & Szegedy, 2015; Ulyanov et al., 2016; Cai et al., 2021). For an elementwise non-linear relu activation, we have the following expression for the forward pass of a single sample $\mathbf{x}_s \in \mathbb{R}^D$ after one layer including layer norm:

$$f(\mathbf{x}_s) = \text{relu}(\text{LN}(\mathbf{x}_s W)) \tag{16}$$

We show that a sample $\mathbf{x}_s$ and $\mathbf{x}_{s'} = \alpha \mathbf{x}_s + \delta$ yield the same output when processed through layer norm following a linear projection.

Let $\overrightarrow{\mu}_s, \overrightarrow{\sigma}_s \in \mathbb{R}^D$ be the statistics used for standardization of a sample s. For instance, $\overrightarrow{\mu}_{\mathbf{x}_s W}, \overrightarrow{\sigma}_{\mathbf{x}_s W}$ denote the statistics for the linear projection of sample $\mathbf{x}_s$. We let $\overrightarrow{\gamma}, \overrightarrow{\beta} \in \mathbb{R}^D$ be the per-feature transformation parameters. Then,

[4]The beginning of the residual and non-residual Deep Sets and the clean path Set Transformer have layer norm immediately after a linear projection of the input.

$$\text{LN}(\mathbf{x}_{s'}W) = \frac{(\alpha\mathbf{x}_s + \delta)W - \overrightarrow{\mu}_{\mathbf{x}_{s'}W}}{\overrightarrow{\sigma}_{\mathbf{x}_{s'}W}} * \overrightarrow{\gamma} + \overrightarrow{\beta} \tag{17}$$

$$= \frac{\alpha\mathbf{x}_sW + \delta W - (\alpha\overrightarrow{\mu}_{\mathbf{x}_sW} + \delta W)}{\alpha\overrightarrow{\sigma}_{\mathbf{x}_sW}W} * \overrightarrow{\gamma} + \overrightarrow{\beta} \tag{18}$$

$$= \frac{\alpha\mathbf{x}_sW - \alpha\overrightarrow{\mu}_{\mathbf{x}_sW}}{\alpha\overrightarrow{\sigma}_{\mathbf{x}_sW}W} * \overrightarrow{\gamma} + \overrightarrow{\beta} \tag{19}$$

$$= \frac{\mathbf{x}_sW - \overrightarrow{\mu}_{\mathbf{x}_sW}}{\overrightarrow{\sigma}_{\mathbf{x}_sW}} * \overrightarrow{\gamma} + \overrightarrow{\beta} = \text{LN}(\mathbf{x}_sW). \tag{20}$$

Since $\text{LN}(\mathbf{x}_{s'}W) = \text{LN}(\mathbf{x}_sW)$, $f(\mathbf{x}_{s'}) = f(\mathbf{x}_s)$, meaning the two samples are indistinguishable at this point in the network. □

We can use the same proof strategy to also show that layer norm and set norm map sets with the same global mean and standard deviation to the same output. The only difference is that we considering a set $\mathbf{x} \in \mathbb{R}^{S\times D}$ rather than a sample $\mathbf{x}_s \in \mathbb{R}^D$. The intuition is the same: since standardization is happening separately on each set, each set's resulting output will be mean zero with unit variance.

Proposition 5. *Set norm is permutation equivariant.*

Proof. Let $\mu, \sigma \in \mathbb{R}$ be the sample mean and variance over all features in the set, $\overrightarrow{\gamma}, \overrightarrow{\beta} \in \mathbb{R}^{S\times D}$ refer to the appropriate repetition of per-feature parameters in the S dimension. Then,

$$\text{SN}(\pi\mathbf{x}) = \frac{\pi\mathbf{x} - \mu}{\sigma} * \overrightarrow{\gamma} + \overrightarrow{\beta}] \tag{21}$$

$$= \pi\Big[\frac{\mathbf{x} - \mu}{\sigma} * \overrightarrow{\gamma} + \overrightarrow{\beta}\Big] \tag{22}$$

$$= \pi\text{SN}(\mathbf{x}). \tag{23}$$

Equation (22) follows from the fact that μ, σ are scalars and $\overrightarrow{\gamma}, \overrightarrow{\beta}$ are equivalent for every sample in the set. □

Proposition 3. *Among all standardizations which are applied separately per set, the per set standardization of set norm results in minimal loss of information.*

Proof. The possible standardizations which occur separately per set are 1. per set, 2. per set per sample, 3. per set per feature, and 4. per set per sample per feature. The last combination (per set per sample per feature) is ill-defined since each standardization operation occurs only over one element. This leaves the first three.

We show that per set per sample and per set per feature standardizations lose more information than per set does. We first show that they lose as much information as per set standardization since the resulting global mean and variance of each output set is 0 and 1 respectively. Let $\widetilde{\mathbf{x}}_{\{S\}} \in \mathbb{R}^{S\times D}$ be the result of a set when each sample is standardized separately, and let $\widetilde{\mathbf{x}}_{\{D\}} \in \mathbb{R}^{S\times D}$ be the result when each feature is standardized separately. We index into these tensors with subscripts s and d.

Then, the global sample mean μ and sample variance σ of the tensors are zero and one respectively:

$$\mu(\widetilde{\mathbf{x}}_{\{S\}}) = \frac{1}{S}\frac{1}{D}\sum_{s=1}^{S}\sum_{d=1}^{D}\widetilde{\mathbf{x}}_{\{S\},sd} = \frac{1}{D}\sum_{d=1}^{D}(\frac{1}{S}\sum_{s=1}^{S}\widetilde{\mathbf{x}}_{\{S\},sd}) = \frac{1}{D}\sum_{d=1}^{D}0 = 0. \tag{24}$$

$$\sigma^2(\widetilde{\mathbf{x}}_{\{S\}}) = \frac{1}{S}\frac{1}{D}\sum_{s=1}^{S}\sum_{d=1}^{D}(\widetilde{\mathbf{x}}_{\{S\},sd} - \mu(\widetilde{\mathbf{x}}_{\{S\}}))^2 = \frac{1}{S}\frac{1}{D}\sum_{s=1}^{S}\sum_{d=1}^{D}(\widetilde{\mathbf{x}}_{\{S\},sd})^2 = \frac{1}{D}\sum_{d=1}^{D}1 = 1. \tag{25}$$

$$\mu(\widetilde{\mathbf{x}}_{\{D\}}) = \frac{1}{S}\frac{1}{D}\sum_{s=1}^{S}\sum_{d=1}^{D}\widetilde{\mathbf{x}}_{\{D\},sd} = \frac{1}{S}\sum_{s=1}^{S}(\frac{1}{D}\sum_{d=1}^{D}\widetilde{\mathbf{x}}_{\{D\},sd}) = \frac{1}{S}\sum_{s=1}^{S}0 = 0. \tag{26}$$

$$\sigma^2(\widetilde{\mathbf{x}}_{\{D\}}) = \frac{1}{S}\frac{1}{D}\sum_{s=1}^{S}\sum_{d=1}^{D}(\widetilde{\mathbf{x}}_{\{D\},sd} - \mu(\widetilde{\mathbf{x}}_{\{D\}}))^2 = \frac{1}{S}\frac{1}{D}\sum_{s=1}^{S}\sum_{d=1}^{D}(\widetilde{\mathbf{x}}_{\{D\},sd})^2 = \frac{1}{S}\sum_{s=1}^{S}1 = 1. \tag{27}$$

We then show that the global mean and variance are the only information lost by per set standardization. Specifically, given the global mean and variance, we can recover the original input $\mathbf{x}$ of the standardization from the output $\widetilde{\mathbf{x}}$:

$$\widetilde{\mathbf{x}}\sigma + \mu = \frac{\mathbf{x} - \mu}{\sigma}\sigma + \mu = \mathbf{x}.$$

Finally, we show that per set per sample and per set per feature standardization each lose additional information beyond the global set mean and variance. Specifically, per set per sample results in zero mean and unit variance for every sample, removing the means and variances of every sample. Per set per feature standardization similarly removes the means and variances of every feature. Therefore, the least amount of information is lost via the per set standardization setting of set norm. □

C THE CLEAN PATH PRINCIPLE

A Residual Pipeline Separated by Linear Layers Can Suffer From Vanishing or Exploding Gradients. While identity residual connections have been shown to prevent gradient vanishing, He et al. (2016a) note that adding a learned scalar weight to the residual connection, i.e. $\mathbf{x}_{\ell+1} = \lambda_\ell \mathbf{x}_\ell + \mathcal{F}(\mathbf{x}_\ell)$, can result in vanishing or exploding gradients if the λ scalars are consistently large (e.g. > 1) or small (< 1). We note that the same is true of a weight matrix in between residual connections, i.e. $\mathbf{x}_{\ell+1} = \mathbf{x}_\ell W + \mathcal{F}((\mathbf{x}_\ell)W)$. By replacing a product of ones inside the gradient expression with a product of learned weights, the addition of weight matrices along the residual path reintroduces the potential for the gradients of early layers in a network to be very large or very small.

Normalization Along The Residual Path Can Also Cause Vanishing or Exploding Gradients. Wang et al. (2019b) note that layer norm along the residual path results in a product of its derivatives accumulating in the gradient computation (i.e. $\prod_{k=\ell}^{L-1}\frac{\partial \text{LN}(y_k)}{y_k}$), with the number of items scaling with depth of the network. The same is true replacing layer norm with another normalization layer along the path. On the contrary, moving the normalization off the path removes this product term from the gradient. Indeed, Xiong et al. (2020) show that transformers without a clean path require careful learning rate scheduling unlike their clean path counterparts, and Wang et al. (2019b) see that for deep transformers, only the clean path variants converge during training.

ReLu Activations Between Connections Force the Output to Grow. Placing a ReLU activation between residual connections, i.e. $\mathbf{x}_{\ell+1} = \text{relu}(\mathbf{x}_\ell) + \mathcal{F}(\text{relu}(\mathbf{x}_\ell))$ ensures that the first additive term is always positive. Such a constraint can cause output explosion for deep models, causing instability.

D EXPERIMENTAL CONFIGURATION

Across experiments and models we purposefully keep hyperparameters consistent to illustrate the easy-to-use (black box) nature of our proposed models. All experiments and models are implemented in PyTorch, the implementation is available at: *deanonymized*.

Table 6: Detailed Deep Sets++ architecture.

Encoder			Aggregation	Decoder
Initial layer	Residual Block $\times 25$	Last block		
FC(128)	SetNorm(128)	SetNorm(128)	Sum/Max	FC(128)
	ReLU	ReLU		ReLU
	FC(128)	FC(128)		FC(128)
	SetNorm(128)			ReLU
	ReLU			FC(128)
	FC(128)			ReLU
	Addition			FC(n_outputs)

Table 7: Detailed Deep Sets architecture.

Encoder		Aggregation	Decoder
MLP block $\times 51$			
FC(128)	FC(128)	Sum/Max	FC(128)
ReLU			ReLU
			FC(128)
			ReLU
			FC(128)
			ReLU
			FC(n_outputs)

D.1 MODELS

All models use 128 hidden units throughout (both encoder and decoder).

Deep Sets and Deep Sets++. The Deep Sets encoder model consists of 51 MLP blocks, each $f(\mathbf{x}) = \text{relu}(\mathbf{x}W + \mathbf{b})$. The encoder ends with a final linear layer with bias, for a total of 52 weight layers.

When we add normalization to Deep Sets (i.e. the results of Table 1), we place the normalization right before the nonlinear activaion and remove the bias term in the affine transform, following Ioffe & Szegedy (2015) and (Ba et al., 2016).

The Deep Sets++ encoder model consists of 25 He residual blocks, each normalization-activation-linear-normalization-activation-linear with a residual connection from start to end of the block. The encoder starts with a full connected layer and ends with a final normalization-activation-linear block. Thus, the number of linear weight matrices in total is 52, matching the Deep Sets model.

All Deep Sets models share the same aggregation (sum, max) and decoder (three MLP blocks and a final output layer). Linear layers preceding a normalization do not have bias terms.

Set Transformer and Set Transformer++. Set Transformer and Set Transformer++ both have 16 induced set attention blocks with 32 inducing points. The encoder of Set Transformer++ terminates with a normalization layer, following Xiong et al. (2020) for the Pre-LN Transformer. The decoder of the Set Transformer and Set Transformer++ architectures consists of three self-attention transformer blocks, followed by the appropriate output layer. We used four heads in the multi-head attention blocks for both encoder and decoder, following (Lee et al., 2019).

The Set Transformer encoder block is called the induced set attention block, or ISAB. ISAB consists of two multihead attention blocks (MAB), not to be confused with multihead attention (MultiheadAttn) as defined in Vaswani et al. (2017):

$$\text{MAB}_K(X, Y) = H + FC(H) \tag{28}$$

$$\text{where } H = XW_Q + \text{MultiheadAttn}_K(X, Y, Y). \tag{29}$$

MAB is similar to the transformer attention block except that the query matrix is used in the residual connection over the multihead attention, rather than the input.

Table 8: Detailed Set Transformer++ architecture.

Encoder	Aggregation	Decoder
FC(128)	PMA(128, 4)	SAB(128, 4)
ISAB++(128, 4, 32) $\times$16		SAB(128, 4)
SetNorm(128)		FC(n_outputs)

Table 9: Detailed Set Transformer architecture.

Encoder	Aggregation	Decoder
FC(128)	PMA(128, 4)	SAB(128, 4)
ISAB(128, 4, 32) $\times$16		SAB(128, 4)
		FC(n_outputs)

Lee et al. (2019) define the induced set attention block ISAB(D, K, M) with D hidden units, K heads and M inducing points as follows:

$$ISAB_M(X) = \text{MAB}_K(X, H) \in \mathbb{R}^{S\times D} \tag{30}$$

$$\text{where } H = \text{MAB}_K(I, X) \in \mathbb{R}^{M\times D}. \tag{31}$$

We define a variant of the ISAB model, which we call ISAB++, that changes the residual connections and adds normalization in the Pre-LN position (Klein et al., 2017; Vaswani et al., 2018; Xiong et al., 2020;?) off the residual path. We define two multi head attention blocks MAB^1 and MAB^2 with K heads as

$$\text{MAB}^1_K(X, Y) = H + \text{FC}(\text{ReLU}(\text{SetNorm}(H))) \tag{32}$$

$$\text{where } H = X + \text{MultiheadAttn}_K(X, \text{SetNorm}(Y), Y). \tag{33}$$

$$\text{MAB}^2_K(X, Y) = H + \text{FC}(\text{ReLU}(\text{SetNorm}(H))) \tag{34}$$

$$\text{where } H = X + \text{MultiheadAttn}_K(\text{SetNorm}(X), \text{SetNorm}(Y), Y). \tag{35}$$

Then, the ISAB++(D,K, M) block with D hidden units, K heads and M inducing points is defined as

$$\text{ISAB++}_M(X) = \text{MAB}^2_K(X, H) \in \mathbb{R}^{S\times D}, \tag{36}$$

$$H = \text{MAB}^1_K(I, X) \in \mathbb{R}^{M\times D}. \tag{37}$$

The reason why MAB^1_K does not include normalization on the first input is because that input (I) is a learned parameter.

Convolutional blocks for set anomaly For our set anomaly task on CelebA, similarly to Zaheer et al. (2017), we add at the beginning of all the considered architectures 9 convolutional layers with 3×3 filters. Specifically, we start 2D convolutional layers with 32, 32, 64 feature-maps followed by max pooling, we follow with 2D convolutional layers with 64, 64, 128 feature maps followed by another max pooling. The last block has 128, 128, 256 2D convolutional layers followed by a max-pooling layer with size 5, for a final input to the permutation invariant models of 255 features. To remain consistent with our general purpose model, we do not modify further the architecture differently from what was done in Zaheer et al. (2017) and Lee et al. (2019).

D.2 Experimental Setup

Hematocrit, Point Cloud and Normal Var use a fixed sample size of 1000. MNIST Var and CelebA use a sample size of 10 due to the high-dimensionality of the images in input. The only architectural differences across experiment, are that for the Deep Sets architecture, we use sum aggregation for all experiments except Point Cloud, where we use max aggregation, following (Zaheer et al., 2017) and the use of convolutional layers for CelebA.

All models are trained with a batch size of 64 for 50 epochs, except for Hematocrit where we train for 30 given the much larger size of the training dataset (i.e. 90k vs. $\leq$ 10k). All results are reported as test MSE (or cross entropy for point cloud) at the last epoch. We did not use early stopping or other model selection techniques, and there was no sign of overfitting. Results are reported setting seed 0,1 and 2 for initialization weights. We use the Adam optimizer with learning rate 1e-4 throughout.

Table 10: Set norm and/or equivariant residual connection does not lead to better results across the batch for Deep Sets.

	Normalization	Hematocrit	Point Cloud	Mnist Var	Normal Var
Deep Sets	none	$\mathbf{19.1257 \pm 0.0361}$	0.7357 ± 0.0119	0.4520 ± 0.0111	$\mathbf{0.0417 \pm 0.0074}$
	set norm	20.4169 ± 0.0164	0.7497 ± 0.0115	$\mathbf{0.4400 \pm 0.0051}$	4.1356 ± 0.1791
Deep Sets++	none	19.3767 ± 0.0270	0.6838 ± 0.0056	0.4968 ± 0.0137	0.3673 ± 0.1098
	set norm	19.5882 ± 0.0555	$\mathbf{0.6703 \pm 0.0093}$	0.5895 ± 0.0114	0.0707 ± 0.0326

Table 11: Set norm and/or equivariant residual connection does not lead to better results across the batch for Set Transformer.

	Normalization	Hematocrit	Point Cloud	Mnist Var	Normal Var
Set Transformer	none	$\mathbf{18.8750 \pm 0.0058}$	0.7487 ± 0.0381	0.6151 ± 0.0072	0.0016 ± 0.0005
	set_norm	18.9810 ± 0.0833	0.7268 ± 0.0186	$\mathbf{0.4890 \pm 0.0028}$	0.0099 ± 0.0032
Set Transformer++	none	19.0822 ± 0.0270	0.6834 ± 0.0109	0.8061 ± 0.0068	$\mathbf{0.0004 \pm 0.0000}$
	set_norm	18.9223 ± 0.0273	$\mathbf{0.6366 \pm 0.0004}$	1.1525 ± 0.0158	0.0050 ± 0.0008

Creating Figure 1. Once every 400 steps for Hematocrit, Point Cloud, and MNIST and once every 100 steps for Normal Var, we plot the loss of the full test set and the average of the loss on the last 20 training batches. We plot epochs on the x-axis and MSE or cross entropy on the y-axis.

E Additional results

E.1 Effect of set norm and equivariant residual connection on shallow architectures

Here, we explore if the addition of set norm and equivariant residual connection has a positive effect also on shallow architectures. Results are reported in Table 10 and Table 11 for Deep Sets/Deep Sets++ and SetTransformer/Set Transformer++ respectively and they show that there is no clear improvement of the additions compared to their conterpart. For Point Cloud, set norm and equivariant residual connection lead to better results. Equivariant residual connection improve the prediction of normal variance for Set Transformer. Set norm by itself improves the prediction of Mnist Var but, on this task, equivariant residual connection degrade results. For hematocrit prediction the original shallow architectures are better than with any other addition.

E.2 Understanding Feature Norm vs. Set Norm for Deep Sets on Point Cloud.

Here, we explore why feature norm performs better than set norm with Deep Sets on Point Cloud. To test whether the difference in performance comes from a benefit of feature norm or an issue of set norm, we test the same experiment on a hybrid normalization: a per set per feature standardization, per feature transformation. We find that the resulting model performs on par with feature norm (0.9550 vs. 0.9309 for feature norm) and better than set norm (1.7334) (see train and test loss curves in Figure 5). This result suggests that standardizing each set separately, as is done by set norm, is not a problem. Instead, the benefit likely comes from the per feature standardization and is a task-specific benefit.

E.3 Understanding ResNet vs. He Pipeline for Deep Sets on Point Cloud.

We explore why Deep Sets with the non clean ResNet residual pipeline performs better on Point Cloud than Deep Sets with the clean He residual pipeline. Specifically, to test whether the difference is due to the relu in between connections, we design another residual pipeline where the connections (i.e. additions) are more frequent and also separated by a relu nonlinearity. We call this pipeline FreqAdd. This new architecture is shown in Table 12 and comparison of loss curves is in Figure 6 where we can observe that the architecture with more residual connection FreqAdd has even better performances than the non-clean pipeline. We speculate that this might be due to peculiarities of Point Cloud which benefit from continual addition positive values. Indeed, in the original Deep Sets paper (Zaheer et al., 2017), the authors add a ReLU to the end of the encoder for the architecture

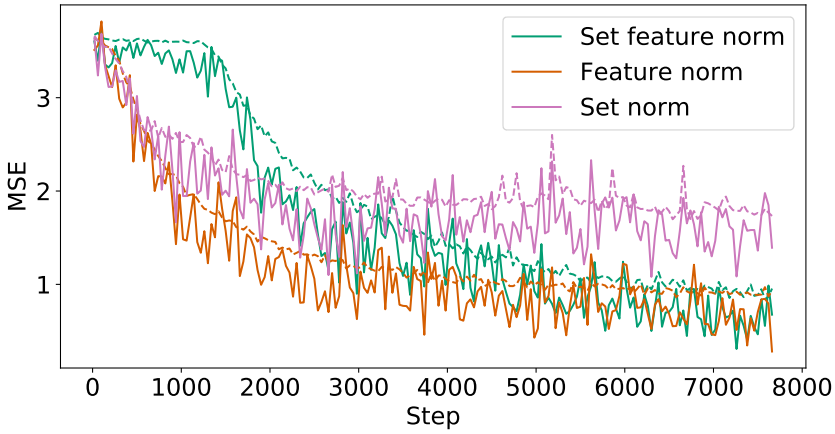


Figure 5: Train (full line) and test (dashed line) loss curves on Point Cloud classification comparing the three normalization. Set feature norm and feature norm performs similarly even if set feature norm removes information about scale and translation from the sets.

Table 12: Detailed DeepSets more residuals architecture.

Encoder		Aggregation	Decoder
Residual block $\times 51$			
FC(128)	FC(128)	Sum/Max	FC(128)
SetNorm(128)			ReLU
Addition			FC(128)
ReLU			ReLU
			FC(128)
			ReLU
			FC(n_outputs)

tailored to point cloud classification, and such a nonlinearity is noticeably missing from the model used for any other task.

E.4 Understanding Layer Norm.

Across the experiments presented in Section 5.3, layer norm seems to often be a poor choice for normalization, including sometimes in a transformer-like architecture. Yet the same normalization

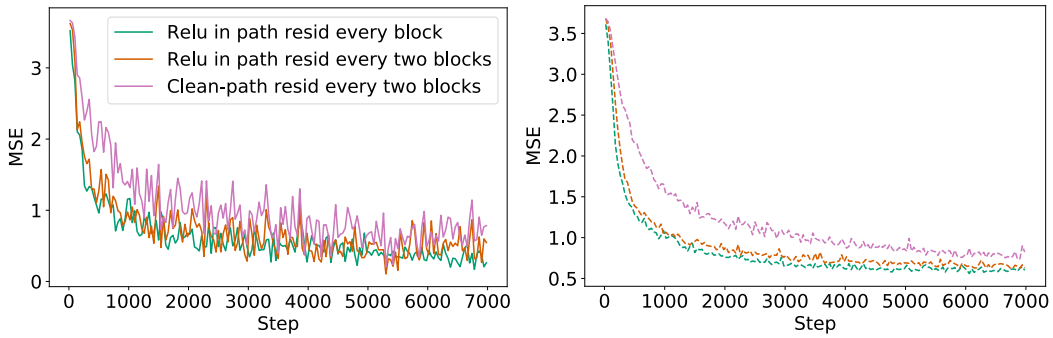
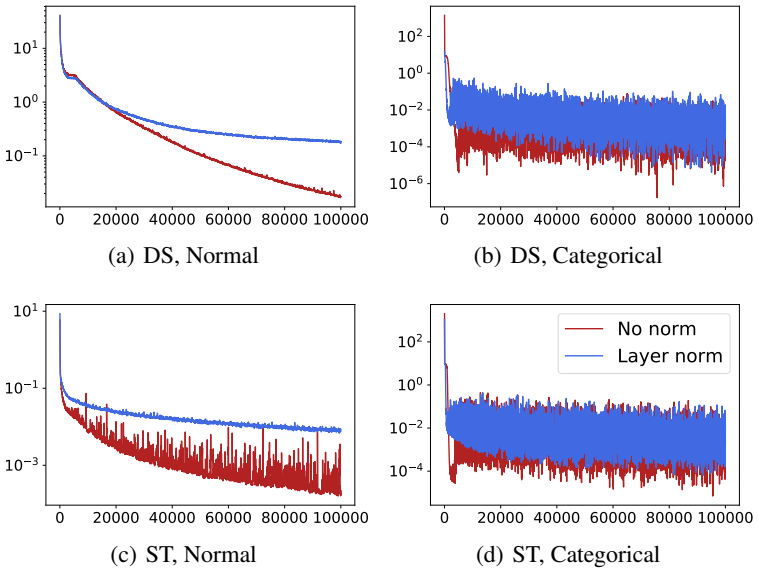


Figure 6: Loss curves for train (left) and test (right) comparing the residual pipelines ResNet (orange), He (magenta) and FreqAdd (green). Adding a positive number more frequently (green) results in better performances for Point Cloud.



(a) DS, Normal (b) DS, Categorical

(c) ST, Normal (d) ST, Categorical

Figure 7: Layer norm is poor choice for tasks with real-valued inputs (a, c), since the normalization will force certain samples to have the same output, limiting representation capacity relative to no normalization. This problem of losing information is less of a concern when the samples (i.e. tokens) are categorical (b, d), as is the case for text data.

has been helpful for many tasks in natural language processing (Ba et al., 2016; Chen et al., 2018). What accounts for the discrepancy?

We note that layer norm's information loss—mapping samples which differ in scale and bias to the same output—is less problematic in the context of text data since the tokens are categorical and thus will not be related via simply a scale and bias. Consequently, their resulting activations are also less likely to be mapped to the same output. In Figure 7, we show that the negative effect of layer norm on a prediction task with real-valued inputs mostly disappears when the inputs become categorical.

E.5 Comparing Point Cloud classification with Task-Specific Models

Here, we compare the performances of DS++ and ST++ unmodified with those of models built specifically for point cloud classification. For a fair comparison, we use the experimental setup and the code provided in SimpleView (Goyal et al., 2021b). In practice, we use their DGCNN-smooth protocol and recorde the test accuracy at 160 epochs to allow models to converge. Sample size for this experiment is the default in the SimpleView repository, 1024. We compared Deep Sets++, Set Transformer++, PointNet++ (Qi et al., 2017b), and SimpleView (Goyal et al., 2021a), as well as the models proposed in the original Deep Sets and Set Transformer papers tailored to point cloud classification, which are different than from the baseline architectures used in our main results. We describe these tailored Deep Sets and Set Transformer models in Table 13 and Table 14.

Results are reported in Table 15 and Figure 8. Deep Sets++ and Set Transformer++ without any modifications both achieve a higher test accuracy than the Deep Sets and Set Transformer models tailor designed for the task. PointNet++ and SimpleView perform best, but both architectures are designed specifically for point cloud classification rather than tasks on sets in general. Concretely, PointNet++ hierarchically assigns each point to centroids using Euclidean distance which is not an informative metric for high-dimensional inputs, e.g. sets of images. SimpleView is a non permutation invariant architecture that represents each point cloud by 2D projections at various angles, such a procedure is ill-suited for sets where samples do not represent points in space.

Table 13: Customized Deep Sets architecture for PointCloud.

Encoder	Aggregation	Decoder
x - max(x)	Max	Dropout(0.5)
FC(256)		FC(256)
Tanh		Tanh
x-max(x)		Dropout(0.5)
FC(256)		FC(n_outputs)
Tanh		
x-max(x)		
FC(256)		
Tanh		

Table 14: Customized Set Transformer architecture for PointCloud.

Encoder	Aggregation	Decoder
FC(128)	Dropout(0.5)	Dropout(0.5)
ISAB(128, 4, 32)	PMA(128, 4)	FC(n_outputs)
ISAB(128, 4, 32)		

Table 15: Point cloud test accuracy

Model	Accuracy
Deep Sets	0.86
Deep Sets++	0.87
Set Transformer	0.86
Set Transformer++	0.87
SimpleView	0.92
PointNet++	0.92

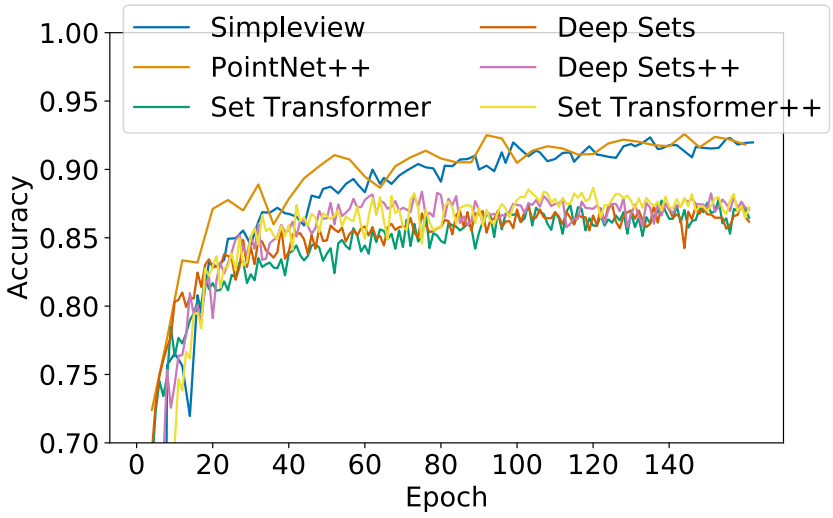


Figure 8: Test accuracy curves of the six architectures on Point Cloud classification.

F GRADIENT COMPUTATION FOR PROPOSED RESIDUAL CONNECTIONS

We compute the gradients for early weights in a deep sets network with our proposed residual connections, equivariant residual connections, mean aggregated residual connections, and max aggregated residual connections.

We denote a single set $\mathbf{x}$ with its samples $\mathbf{x}_1, ...\mathbf{x}_S$. We denote hidden layer activations as $\mathbf{z}_{\ell,s}$ for layer ℓ and sample s. In the case of no residual connection, $\mathbf{z}_{\ell,s} = \text{ReLu}(\mathbf{z}_{\ell-1,s}W_\ell + b_\ell)$. We denote the output after an L-layer encoder and permutation invariant aggregation as $\mathbf{y} = \sum_s \mathbf{z}_{L,s}$ (we use sum for illustration but note that our conclusions are the same also for max). For simplicity let the hidden dimension remain constant throughout the encoder.

Now, we can write the gradient of weight matrix of the first layer W_1 as follows:

$$\frac{\partial \mathcal{L}}{\partial W_1} = \frac{\partial \mathcal{L}}{\partial W_1} \sum_s \frac{\partial \mathbf{y}}{\partial \mathbf{z}_{L,s}} \frac{\partial \mathbf{z}_{L,s}}{\partial W_1}. \tag{38}$$

For each choice of residual connections in the encoder (none, equivariant, mean aggregated, max aggregated), the only term which differs is $\frac{\partial \mathbf{z}_{L,s}}{\partial W_1}$.

For no residual connections, $\frac{\partial \mathbf{z}_{L,s}}{\partial W_1} = \frac{\partial \mathbf{z}_{L,s}}{\partial \mathbf{z}_{1,s}} \frac{\partial \mathbf{z}_{1,s}}{\partial W_1}$, where the first term is a product of terms which can become vanishingly small:

$$\frac{\partial \mathbf{z}_{L,s}}{\partial \mathbf{z}_{1,s}} = \prod_{\ell=2}^{L} \frac{\partial \mathbf{z}_{\ell,s}}{\partial \mathbf{z}_{\ell-1,s}} \tag{39}$$

$$= \prod_{\ell=2}^{L} \frac{\partial \text{ReLU}(\mathbf{z}_{\ell,s})}{\partial \mathbf{z}_{\ell,s}} W_\ell. \tag{40}$$

Equivariant residual connections prevent vanishing gradients by passing forward the result of the previous computation along that sample's path, i.e. $\mathbf{z}_{\ell,s} = \text{ReLu}(\mathbf{z}_{\ell-1,s}W_\ell + b_\ell) + \mathbf{z}_{\ell-1,s}$:

$$\frac{\partial \mathbf{z}_{L,s}}{\partial \mathbf{z}_{1,s}} = \prod_{\ell=2}^{L} \frac{\partial \mathbf{z}_{\ell,s}}{\partial \mathbf{z}_{\ell-1,s}} \tag{41}$$

$$= \prod_{\ell=2}^{L} \frac{\partial \text{ReLU}(\mathbf{z}_{\ell,s})}{\partial \mathbf{z}_{\ell,s}} (1 + W_\ell). \tag{42}$$

Aggregated residual connections pass the output of the previous layer, aggregated over all the samples in that layer, on a per feature basis. As a result, there is not just one path from $\mathbf{z}_{L,s}$ to W_1 (i.e. through previous $\mathbf{z}$ at the same sample index s), but rather S^{L-1} paths going through all the samples at every layer. Let P denote the set of paths from $\mathbf{z}_{L,s}$ to $\mathbf{z}_{1,s}$. Let the main path go from one sample to itself $\mathbf{z}_{\ell,s}$ to $\mathbf{z}_{\ell-1,s}$ at each step, and let the auxiliary paths consist of the effect of that samples to the remaining samples, e.g. $\mathbf{z}_{\ell,s}$ to $\mathbf{z}_{\ell-1,s'}$, $s \neq s'$.

For a mean aggregated residual, we have $\mathbf{z}_{\ell,s} = \text{ReLu}(\mathbf{z}_{\ell-1,s}W_\ell + b_\ell) + \frac{1}{S}\sum_{s'=1}^{S} \mathbf{z}_{\ell-1,s'}$. The gradient for a single time step on the main path is $d_m = \frac{\partial \mathbf{z}_{\ell,s}}{\partial \mathbf{z}_{\ell-1,s}} = (\frac{1}{S} + W_\ell)$. The gradient for a single time step on an auxiliary path is $d_a = \frac{\partial \mathbf{z}_{\ell,s}}{\partial \mathbf{z}_{\ell-1,s'}} = \frac{1}{S}$. The gradient computation of $\frac{\partial \mathbf{z}_{L,s}}{\partial W_1}$ takes into account all paths, which we can decompose into the full main path, the many full auxiliary paths, and paths that are combination of both through the layers.

In the remaining text we remove the ReLU gradient terms in the expression below for readability. Note that, ReLU gradient terms simply serve to zero out a particular gradient path if the output $\mathbf{z}_{\ell,s}$ was squashed to zero by the ReLU function.

For each sample we have $\frac{\partial \mathbf{z}_{L,s}}{\partial W_1} = \sum_{p_s \in P} C_{p_s} \frac{\partial \mathbf{z}_{1,s}}{\partial W_1} = \sum_{p_s \in P} C_{p_s} \mathbf{x}_s$. The constant C_{p_s} is based on the chosen path through the network and represents a product of the terms d_m or d_a depending on

the chosen path across the network. While different paths will be multiplied by a different sample at the end (i.e. $\mathbf{x}_s$ for path p_s and $\mathbf{x}_t$ for path p_t), we consider $\sum_{p_s \in P} C_{p_s}$ below to illustrate why the gradients do not vanish with more layers (ignoring ReLU terms):

$$\sum_{p_s \in P} C_{p_s} = \prod_{\ell=2}^{L} (\frac{1}{S} + W_\ell) + (S-1)^{L-1} (\frac{1}{S})^{L-1} + (S-1)^{L-2} \sum_{\ell=2}^{L} (\frac{1}{S} + W_\ell)(\frac{1}{S})^{L-2} \tag{43}$$

$$+ (S-1)^{L-3} \sum_{\ell_1=2}^{L} \sum_{\ell_2=\ell_1}^{L} (\frac{1}{S} + W_{\ell_1})(\frac{1}{S} + W_{\ell_2})(\frac{1}{S})^{L-3} + ... \tag{44}$$

$$+ (S-1) \sum_{\ell_1=2}^{L} \sum_{\ell_2=\ell_1}^{L} ... \sum_{\ell_{L-2}=\ell_{L-3}}^{L} (\frac{1}{S} + W_{\ell_1})(\frac{1}{S} + W_{\ell_2})...(\frac{1}{S} + W_{\ell_{L-2}})(\frac{1}{S}). \tag{45}$$

This expression is a summation of parameter terms A and non-parameter terms B, i.e. $\sum_{p_s \in P} C_{p_s} = A + B$. The non-parameter terms can be written as follows:

$$B = \Big(\frac{1}{S}\Big)^{L-1} + \binom{L-1}{1}\Big(\frac{1}{S}\Big)^{L-2}\Big(\frac{S-1}{S}\Big) + \binom{L-1}{2}\Big(\frac{1}{S}\Big)^{L-3}\Big(\frac{S-1}{S}\Big)^{2} + ... + \Big(\frac{S-1}{S}\Big)^{L-1}.$$

This results in:

$$B = (\frac{1}{S})^{L-1}\Big[1 + \binom{L-1}{1}(S-1) + \binom{L-1}{2}(S-1)^2 + ... + (S-1)^{L-1}\Big] \tag{46}$$

$$= (\frac{1}{S})^{L-1} \sum_{k=0}^{\infty} \binom{L-1}{k} (S-1)^k \tag{47}$$

$$= (\frac{1}{S})^{L-1}\big(1 + (S-1)\big)^{L-1} = 1. \tag{48}$$

Regardless of the depth of the network L, $\sum_{p_s \in P} C_{p_s} = A + B = A + 1$, suggesting that mean aggregated residuals can also help with vanishing gradients.

We can perform an analogous analysis with max aggregated residuals, i.e. $\mathbf{z}_{\ell,s} = \text{ReLu}(\mathbf{z}_{\ell-1,s} W_\ell + b_\ell) + \max_{s'} \mathbf{z}_{\ell-1,s'}$, where $\max_{s'} \mathbf{z}_{\ell-1,s'}$ is computed on a per-feature basis. The primary insight that for a given feature, there will be at most two paths to take into account, the main path and a single max auxiliary path. The max path is the path through the sample with the maximum feature activation at each layer. Each feature max path can be different, and we index this via $\mathbf{z}_{\ell,s_{\max}}$ where $s_{\max}$ indexes each feature dimension separately.

$$\frac{\partial \mathbf{z}_{L,s}}{\partial W_1} = \prod_{\ell=2}^{L} (\frac{\partial \mathbf{z}_{\ell,s}}{\partial \mathbf{z}_{\ell-1,s}}) \frac{\partial \mathbf{z}_{1,s}}{\partial W_1} + \frac{\partial \mathbf{z}_{L,s}}{\partial \mathbf{z}_{L-1,s_{\max}}} ... \frac{\partial \mathbf{z}_{2,s_{\max}}}{\partial \mathbf{z}_{1,s_{\max}}} \frac{\partial \mathbf{z}_{1,s_{\max}}}{\partial W_1}. \tag{49}$$

Ignoring the ReLU gradient terms again, the gradient for a main path step only is W_ℓ. The gradient for a max auxiliary path step is 1, and the gradient when a step is both the main and a max path is $1 + W_\ell$.

At the one extreme, if the main and max path completely overlap, then the gradient is equivalent to the equivariant case for that single sample $\mathbf{z}_{L,s}$. On the other hand, if the main and max paths do not overlap at all, then the first term in Equation (49) can get vanishingly small but the second term will not (the gradient for the path will be $\mathbf{x}_{s_{\max}}$). Many other in-between options exist which substitute in $1 + W_\ell$ factors for the main path term in Equation (49), and these options strictly improve the robustness of the first term against vanishing gradients.