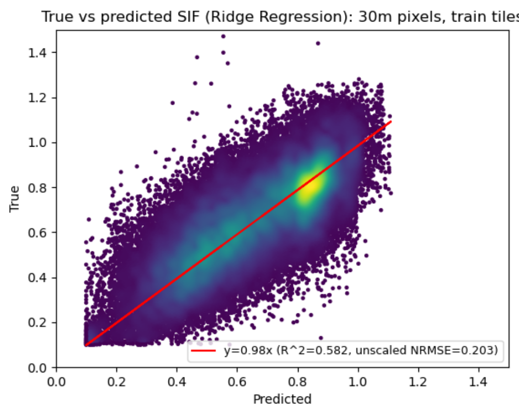
Appendices for *Monitoring Vegetation From Space at Extremely Fine Resolutions via Coarsely-Supervised Smooth U-Net*

A Additional Results

We break down the results on "train tiles" by land cover type (Table 3) and resolution (Table 4). CS-SUNet performs best across all land cover types and resolutions, indicating the robustness of our approach.

Figure 4 presents scatterplots of true vs. predicted SIF for fine-resolution (30m) pixels, for CS-SUNet and the best performing baseline (Ridge Regression). CS-SUNet's predictions are generally closer to the identity function, indicating more accurate predictions.



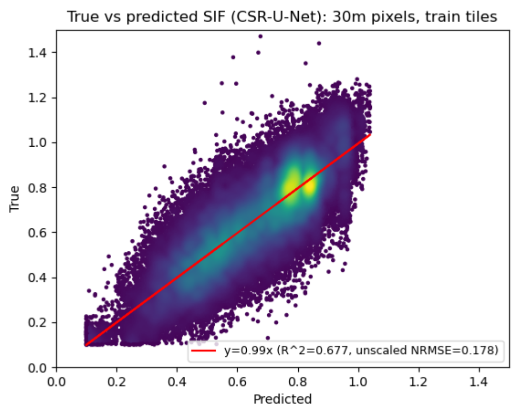


Figure 4: Ground-truth vs. predicted SIF for 30m pixels, on train tiles. **Top:** Ridge Regression, **Bottom:** CS-SUNet

B Importance of Regularization

As described in the paper, we find that early stopping (based on a small fine-resolution validation set) and/or using the smoothness loss is critical to producing reasonable results. Without regularization, the model can overfit in a way that is unique to coarsely-supervised regression tasks. For example, Figure 5 plots losses over time for a model run that is not sufficiently regularized. Note that the coarse-resolution validation loss (green) decreases continuously until around epoch 60, but the fine-resolution losses (red/orange) start increasing after epoch 20-30. In later epochs, the model is making pixel predictions that produce the correct tile average SIF, but are inaccurate for the individual pixels.

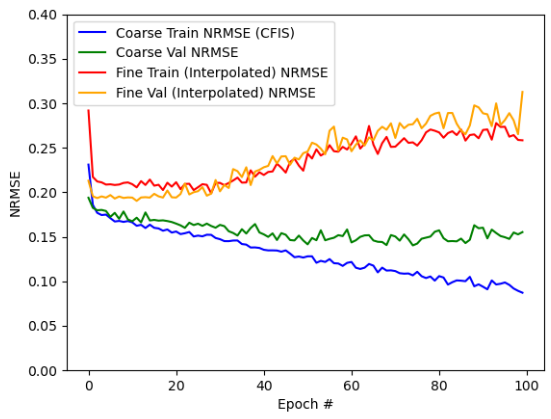


Figure 5: Losses over time in a model that is not sufficiently regularized. Note that the coarse-resolution losses (blue/green) continue decreasing for while, but the fine-resolution losses (red/orange) go up quickly after epoch 20-30, indicating overfitting to the coarse labels.

Specifically, we observed that over-trained models tend to output extreme maps; the model learns how to keep pushing the predictions for low SIF regions down and high SIF regions up, in a way that maintains the correct average. An example of this is shown in Figure 6.

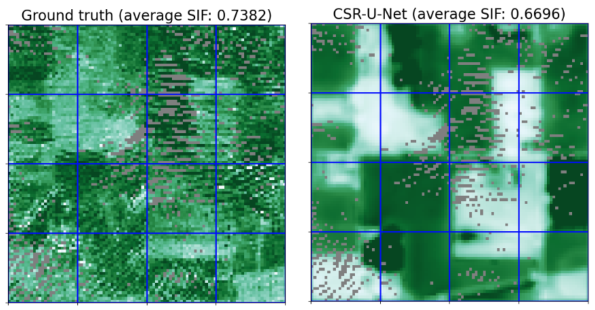

Ground truth (average SIF: 0.7382)
CSR-U-Net (average SIF: 0.6696)

Figure 6: Example of overfitting. **Left:** ground-truth SIF map. **Right:** prediction by a U-Net that is over-trained. Note that the average tile SIFs are not too different, but the model tends to output extremely low and high values that do not reflect reality.

Given the coarse nature of training labels, the only way for the model to avoid this is to look across multiple tiles, and ensure that pixels with similar features in different tiles have similar SIF predictions. In other words, the model needs to be well-regularized. So far, we found that early stopping (based on a fine-resolution validation set) and a smoothness loss are the most effective forms of regularization for our problem. Early stopping is known to implicitly encourage smooth mod-

Method	**Grassland**	**Corn**	**Soybean**	**Deciduous Forest**
Trivial: predict coarse	*0.255*	*0.225*	*0.278*	*0.196*
Ridge Regression	0.230	0.183	0.212	0.192
Gradient Boosting	0.223	0.198	0.219	0.206
Random Forest	0.237 ± *0.002*	0.212 ± *0.001*	0.221 ± *0.003*	0.211 ± *0.003*
ANN	0.247 ± *0.005*	0.193 ± *0.009*	0.226 ± *0.007*	0.193 ± *0.006*
Pixel NN	0.219 ± *0.002*	0.189 ± *0.006*	0.217 ± *0.004*	0.207 ± *0.004*
Vanilla U-Net	0.211 ± *0.010*	0.208 ± *0.008*	0.258 ± *0.011*	0.179 ± *0.002*
CS-SUNet	**0.188** ± *0.007*	**0.167** ± *0.003*	**0.204** ± *0.008*	**0.171** ± *0.003*

Table 3: NRMSE by land cover type, 30m pixels in train tiles (where 3km coarse-resolution labels were seen during training). Lower is better.

Method	**30m**	**90m**	**150m**	**300m**	**600m**
Trivial: predict coarse	*0.248*	*0.245*	*0.229*	*0.203*	*0.165*
Ridge Regression	0.203	0.177	0.155	0.127	0.096
Gradient Boosting	0.212	0.183	0.163	0.134	0.105
Random Forest	0.220 ± *0.002*	0.184 ± *0.001*	0.163 ± *0.000*	0.135 ± *0.000*	0.111 ± *0.000*
ANN	0.215 ± *0.005*	0.186 ± *0.002*	0.164 ± *0.002*	0.133 ± *0.002*	0.108 ± *0.006*
Pixel NN	0.208 ± *0.004*	0.179 ± *0.001*	0.158 ± *0.001*	0.130 ± *0.001*	0.099 ± *0.002*
Vanilla U-Net	0.223 ± *0.005*	0.194 ± *0.001*	0.176 ± *0.002*	0.150 ± *0.002*	0.121 ± *0.003*
CS-SUNet	**0.184** ± *0.005*	**0.168** ± *0.002*	**0.150** ± *0.001*	**0.123** ± *0.000*	**0.094** ± *0.001*

Table 4: NRMSE by resolution, train tiles (where 3km coarse-resolution labels were seen during training). Lower is better.

els where similar inputs map to similar outputs [Rosca *et al.*, 2020].

C Impact of Smoothness Loss

While early stopping already does a lot to regularize the model, we find that incorporating a smoothness loss can further prevent the model from overfitting in later epochs, as it explicitly penalizes overfitted models (it places a large penalty if pixels with similar input features map to very different SIF). As shown in Figure 7, having a smoothness loss term tends to stabilize the fine-resolution losses later in the training process, and prevents them from getting much worse.

We then investigate varying the strength of the smoothness loss λ (fixing $\tau = 0.5$); the results are shown in Table 5. When $\lambda = 0$ (no smoothness loss), the performance is already quite good, likely because early stopping is enough to yield a well-regularized model. Increasing λ can slightly improve performance on the fine-resolution validation set, and we find that the model performs well over a wide range of λ up to 2. When λ gets too high, model performance degrades – having too much weight on the smoothness loss encourages the model to output similar predictions everywhere.

Finally, we check how robust our model is to changes in τ, the "spread parameter" in the smoothness kernel (fixing $\lambda = 0.5$). The results are shown in Table 6. The model performs well over a wide range of τ, from 0.1 to 100. A high value of τ means that the similarity function decays towards 0 quickly, so it is rare for pixels to be considered similar; the effect is similar to removing the smoothness loss.

λ	**NRMSE** (fine val pixels)
0	0.185
0.01	0.186
0.1	0.184
0.3	0.182
0.5	**0.182**
0.7	0.182
1	0.182
2	0.187
5	0.197
10	0.208
100	0.261

Table 5: Impact of changing λ (smoothness loss weight)

τ	**NRMSE** (fine val pixels)
0.01	0.213
0.1	0.187
0.2	0.183
0.3	0.182
0.5	**0.182**
0.7	0.182
1	0.183
10	0.185
100	0.186

Table 6: Impact of changing τ (spread parameter in smoothness loss)

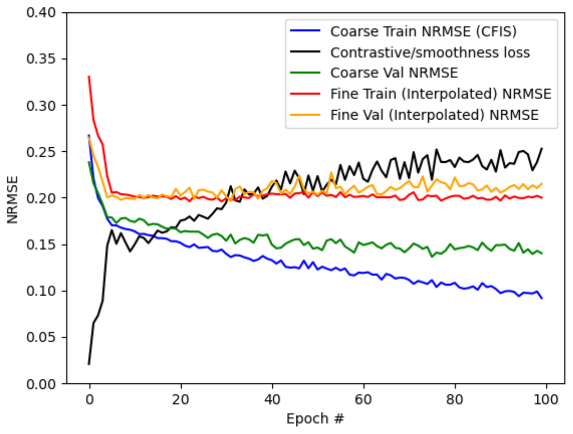


Figure 7: Losses over time with smoothness loss. Note that the red/orange lines are a lot flatter in later epochs, indicating that the fine-resolution performance does not degrade as much due to overfitting.

D Rationale Behind Smoothness Loss

To motivate the design of the smoothness loss, we sample random pairs of pixels of each land cover type. At different ranges of input (reflectance) similarity, we show the distribution of SIF differences in Figure 8. These plots are for soybean, but the trend holds generally.

For pixels with low input similarity (top left), SIF varies a lot. But for pixels with high input siimlarity (bottom right), the differences in SIF are much smaller. This demonstrates that pixels that are similar in input features are also likely to have similar SIFs. Thus, for pixels with similar input features, the model should be penalized if it outputs SIF predictions that are too different – this indicates overfitting.

E Dataset Summary

Table 7 summarizes the input features used, and Table 8 summarizes the SIF datasets used. For SIF data, we use the CFIS dataset [Frankenberg *et al.*, 2018] for fine-resolution measurements (as well as aggregated coarse-resolution measurements), and the OCO-2 dataset [Sun *et al.*, 2017] for additional coarse-resolution measurements.

Dataset	# vars
Landsat surface reflectance Blue, green, red, near infrared, etc.	8
FLDAS land data Rainfall, temperature, radiation	3
CDL land cover types (binary masks) Corn, soybean, grassland, forest, etc.	11

Table 7: Summary of input features used

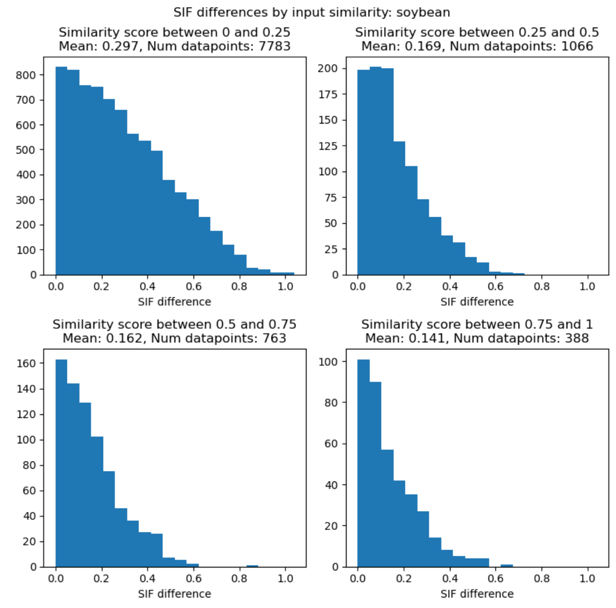


Figure 8: Distributions of SIF differences, between pixels of varying similarity levels

Dataset	Resolution	# 3km labels in train set
CFIS	30 m	712
OCO-2	3 km	1390

Table 8: Summary of SIF datasets used

F List of Features

Here is a full list of input features used in our tiles. We downloaded Landsat and Cropland Data Layer features from Google Earth Engine, and the FLDAS features from NASA Earth Data portal.[2] Each 30m pixel has a value for each of these features.

Landsat surface reflectance [Masek *et al.*, 2006]:

1. Ultra blue surface reflectance (435-451 nm)
2. Blue surface reflectance (452-512 nm)
3. Green surface reflectance (533-590 nm)
4. Red surface reflectance (636-673 nm)
5. Near infrared surface reflectance (851-879 nm)
6. Shortwave infrared 1 surface reflectance (1566-1651 nm)
7. Shortwave infrared 2 surface reflectance (2107-2294 nm)
8. Missing reflectance mask (1 if reflectance data is missing, e.g. due to cloud cover)

FLDAS land data features [McNally *et al.*, 2017]:

[2]https://disc.gsfc.nasa.gov/datasets/FLDAS_NOAH01_C_GL_M_001/summary?keywords=%22MERRA-2%22

1. Rainfall flux (kg m-2 s-1)
2. Surface downward shortwave radiation (W m-2)
3. Surface air temperature (K)

Cropland Data Layer land cover types [NASS, 2016]:

1. Grassland/pasture
2. Corn
3. Soybean
4. Deciduous Forest
5. Evergreen Forest
6. Developed/Open Space
7. Woody Wetlands
8. Open Water
9. Alfalfa
10. Developed/Low Intensity
11. Developed/High Intensity

G Data Filtering

We filter our dataset to exclude pixels and tiles with insufficient or noisy data. For CFIS SIF labels, geographic coverage is extremely limited (as the measurements were taken from an airplane), so most tiles only have SIF labels for some pixels. As a geographic coverage requirement, we only include 3×3 km tiles that contain at least 1000 pixels with at least 1 CFIS measurement. Our models are trained to predict the average SIF over the pixels with CFIS data, rather than over all pixels within the tile.

At evaluation time, we compare our algorithms' fine-resolution SIF predictions with the ground-truth CFIS SIF labels at different resolutions, including $\{30, 90, 150, 300, 600\}$ meters. To reduce measurement noise in the fine-resolution SIF labels, we only evaluate on small pixels that have at least 30 soundings (observations) and have SIF > 0.1 (because low SIF values are difficult to measure accurately). In the next section, we show the impact of varying the number of soundings; the more soundings a pixel has, the more reliable the SIF label should be, since random noise is reduced through averaging. At resolutions of greater than 30 meters, we also require that 90% of the 30m pixels within the larger pixel have at least 1 CFIS measurement.

For OCO-2 SIF, we remove tiles that have less than 3 soundings, and tiles with SIF below 0.1.

Also, we remove tiles where more than 50% Landsat pixels are missing or unreliable (as defined by the Landsat QA band), since in this case there is not enough data to accurately predict the entire-tile SIF. (Also, Landsat pixels that are near cloudy areas tend to be noisy.) We removed tiles where less than 50% of the tile is covered by one of the common land cover types we use (that make up more than 1% of our dataset).

Soundings	**Ridge NRMSE** (fine train pixels)	**CS-SUNet NRMSE** (fine train pixels)
1	0.261	0.242
5	0.248	0.230
10	0.241	0.223
20	0.212	0.190
30	0.203	0.178

Table 9: Impact of changing the minimum number of soundings.

H Impact of data quality (number of soundings)

As SIF observations are noisy and contain a lot of measurement error, we analyze the impact of this by comparing performance against the minimum number of soundings per pixel. If a pixel has more soundings (observations), its noise should be reduced through averaging (due to the Central Limit Theorem). The results are shown in Table 9.

As we increase the number of soundings and thus reduce measurement error, performance steadily improves, indicating that data noise is a significant issue in evaluation. We mitigate this by only evaluating on pixels with at least 30 soundings to reduce noise in the labels, but our results may still be impacted by data noise.

I Training Details

We train on one NVIDIA Tesla V100 GPU with 16GB memory, on the Linux CentOS 7 operating system. For CS-SUNet, training a model for 100 epochs takes roughly 45 minutes. We used the following libraries with Python 3.7: Matplotlib 3.3.4, Numpy 1.18.1, Pandas 1.1.3, PyTorch 1.7.0, Scikit-Learn 0.24.1, Scipy 1.4.1. For all methods that involve randomness, we report the average and standard deviation using three random seeds: $\{0, 1, 2\}$.

For the baseline methods, we did a grid search over hyperparameters, and chose the configuration that performed best on the fine-resolution validation set. For Ridge Regression, we selected the regularization parameter α from $\{0.01, 0.1, 1, 10, 100, 1000, 10000\}$; we chose $\alpha = 100$. For Gradient Boosting Regressor, we selected the maximum number of iterations from $\{100, 300, 1000\}$, and the maximum depth of the tree from $\{2, 3, None\}$. We chose 100 iterations and max depth of 2. For Random Forest, we selected the number of trees from $\{10, 30, 100, 300, 1000\}$ and the max features per split from $\{2, 5, None\}$; we selected 300 trees, and set max features to 5. For the fully-connected artificial neural network, we selected hidden layer sizes from $\{(100), (20, 20), (100, 100), (100, 100, 100)\}$, initial learning rate from $\{10^{-2}, 10^{-3}, 10^{-4}\}$, and set the maximum number of iterations to 10,000. We chose hidden layer sizes of $(100, 100, 100)$ and an initial learning rate of 0.001.

For CS-SUNet, Pixel NN, and Vanilla U-Net methods, we used the AdamW optimizer and a batch size of 128. Then we did a hyperparameter search, considering learning rates from {1e-4, 3e-4, 1e-3} and weight decay from {0, 1e-4, 1e-3}. We only tuned based on random seed 0. For Pixel NN, as well as "U-Net fine supervision", we chose learning rate 1e-3 and weight decay 1e-3. For the Vanilla U-Net, we chose learning rate 1e-4 and weight decay 0. (Note that for the Vanilla U-Net, we used the model at epoch 100; we did not use early stopping.) For CS-SUNet, we chose learning rate 3e-4 and weight decay 1e-4.

For CS-SUNet's smoothness loss, we considered values of τ (spread) from $\{0.01, 0.1, 0.2, 0.3, 0.5, 0.7, 1, 10, 100\}$ and λ (weight) from $\{0, 0.01, 0.1, 0.3, 0.5, 0.7, 1, 2, 5, 10, 100\}$. We found that $\tau = 0.5$ and $\lambda = 0.5$ gave the best result out of the combinations we tried, although there were many similarly good options. (Coincidentally, setting τ and λ to be equal seems to work well.)

In terms of model architecture, we used a smaller version of U-Net with 2 downsampling and 2 upsampling blocks, with {64, 128, 256} hidden units. We start with a 1x1 convolution for a pixel encoder, followed by a rectified linear unit (ReLU), and then 2 downsampling blocks. Each downsampling block consists of the following sequence: (2×2 average pooling, convolutional layer with filter size 3, ReLU, convolutional layer with filter size 1, ReLU). Note that reducing the second convolutional layer's filter size to 1 reduces the receptive field of each pixel and ensures better localization.

We use 2 upsampling blocks; each involves upsampling the feature map, and then the sequence: (convolutional layer with filter size 3, ReLU, convolutional layer with filter size 1, ReLU). Finally, we concatenate the output feature map with the feature map from the higher-resolution layer in the contracting path.

J Effect of batch normalization

Overall, we found that batch normalization actually made results slightly worse, and made training more unstable. To confirm this, we tried multiple learning rates with and without batch normalization. We used a larger batch size of 256 to make the batch statistics more stable. Even still, removing batch normalization improved performance. Batch normalization does allow us to use higher learning rates [Bjorck *et al.*, 2018], but this does not improve results.

Learning rate	**No batch norm** (fine val NRMSE)	**With batch norm** (fine val NRMSE)
1e-4	**0.183**	0.204
1e-3	0.183	0.193
1e-2	0.194	0.204
1e-1	0.310	0.204

Table 10: Effect of batch normalization

We hypothesize that this is because our problem actually depends on the absolute intensity values of the input images. Batch normalization introduces significant noise by scaling by the mean and standard deviation *of each batch*, which removes information about the raw intensities. This phenomenon is also reported in a blog post.[3] This is fine for tasks with natural images, which tend to be more invariant to shifts in light intensity and color. However, when monitoring vegetation, the absolute intensities matter; for example, traditional vegetation indices are simple mathematical formulas based on the absolute intensity values of different channels in these remote sensing images [Bannari *et al.*, 1995].

[3]https://towardsdatascience.com/pitfalls-with-dropout-and-batchnorm-in-regression-problems-39e02ce08e4d

K Data and Code Availability

We intend to submit an expanded version of this paper to a journal, such as in the field of remote sensing. After the journal article is published, we plan to make the entire dataset and codebase publicly available. All of the raw data used already comes from publicly available sources.

L Evaluation Metrics

We evaluate our model on fine-resolution pixels with at least 30 soundings. We use two standard regression metrics: normalized RMSE and R^2.

The RMSE is the square root of the mean squared error between the prediction and the true value:

$$RMSE = \sqrt{\frac{\sum_i (y_i - \widehat{y}_i)^2}{N}}$$

where y_i is the true SIF for pixel i, $\widehat{y_i}$ is the model's predicted SIF for pixel i, and N is the total number for pixels in the evaluation set. In this paper, we further divide RMSE by the average SIF across the train dataset, to get normalized RMSE (NRMSE).

R^2 is a measure of how much the variation in the data can be explained by the model predictions. Formally,

$$R^2 = 1 - \frac{\sum_i (y_i - \widehat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

where $\bar{y}$ is the average SIF across the entire test dataset. The top of the fraction is the sum of the squared residuals (difference between true SIF and model prediction). The bottom is the total sum of squares (of the difference between the true SIF and the average SIF across the test dataset), which is proportional to the overall variance of the test data.

Supplementary References

[Bannari *et al.*, 1995] A Bannari, D Morin, F Bonn, and AjRsr Huete. A review of vegetation indices. *Remote sensing reviews*, 13(1-2):95–120, 1995.

[Bjorck *et al.*, 2018] Nils Bjorck, Carla P Gomes, Bart Selman, and Kilian Q Weinberger. Understanding batch normalization. *Advances in neural information processing systems*, 31, 2018.

[Frankenberg *et al.*, 2018] Christian Frankenberg, Philipp Köhler, Troy S Magney, Sven Geier, Peter Lawson, Mark Schwochert, James McDuffie, Darren T Drewry, Ryan Pavlick, and Andreas Kuhnert. The chlorophyll fluorescence imaging spectrometer (cfis), mapping far red fluorescence from aircraft. *Remote sensing of environment*, 217:523–536, 2018.

[Masek *et al.*, 2006] Jeffrey G Masek, Eric F Vermote, Nazmi E Saleous, Robert Wolfe, Forrest G Hall, Karl Fred Huemmrich, Feng Gao, Jonathan Kutler, and Teng-Kui Lim. A landsat surface reflectance dataset for north america, 1990-2000. *IEEE Geoscience and Remote Sensing Letters*, 3(1):68–72, 2006.

[McNally *et al.*, 2017] Amy McNally, Kristi Arsenault, Sujay Kumar, Shraddhanand Shukla, Pete Peterson, Shugong Wang, Chris Funk, Christa D Peters-Lidard, and James P Verdin. A land data assimilation system for sub-saharan africa food and water security applications. *Scientific data*, 4(1):1–19, 2017.

[NASS, 2016] USDA NASS. Usda national agricultural statistics service cropland data layer. *USDA-NASS, Washington, DC*, 2016.

[Sun *et al.*, 2017] Ying Sun, Christian Frankenberg, Jeffery D Wood, DS Schimel, Martin Jung, Luis Guanter, DT Drewry, Manish Verma, Albert Porcar-Castell, Timothy J Griffis, et al. Oco-2 advances photosynthesis observation from space via solar-induced chlorophyll fluorescence. *Science*, 358(6360):eaam5747, 2017.