Supplementary Material
Your Local GAN: Designing Two Dimensional Local Attention Mechanisms for Generative Models

Giannis Daras
National Technical University of Athens
daras.giannhs@gmail.com

Augusts Odena
Google Brain
augustusodena@google.com

Han Zhang
Google Brain
zhanghan@google.com

Alexandros G. Dimakis
UT Austin
dimakis@austin.utexas.edu

1. A closer look to our inversion method

In the paper, we presented a novel inversion technique that uses the attention layer of the Discriminator to extract weights for our loss function. This section aims to explain technical details of our implementation and clarify the details of our approach.

We begin with a recap of our method as presented in the paper. Given a real image we pass it to the discriminator and we extract the attention map from the attention layer. This attention map contains for every point of the query image, a probability distribution over the pixels of the key image. We can then convert this attention map to a saliency map: by averaging the attention each key point gets from all the query points, we can get a probability distribution over the "importance" of the pixels of the key image. We denote this saliency map with S. Our proposed inversion algorithm is to perform gradient descent to minimize the discriminator embedding distance, weighted by this salience map:

$$\|\big(D^0(G(z)) - D^0(x)\big) \cdot S'\|^2, \tag{1}$$

where S' is a projected version of saliency map S, x is the image, and D^0 is the Discriminator network up to, but not including, the attention layer.

1.1. Multiple heads and saliency map

There are some practical considerations that we need to address before illustrating that our inversion method indeed works: the most important of which is how the saliency map S looks like.

In our analysis of the YLG attention layers, we explain that because of the Full Information property, our patterns are able, potentially, to discover a dependency between any two pixels of an image. If that is true, we should expect that in the general case our saliency map, generated by the average of all heads, allocates non-zero weights to all image pixels. The important question becomes whether this joint saliency map weights more the pixels that are important for a visually convincing inversion. For example, in case of a bird flying with a blue-sky in the background, we should be ready to accept a small error in some point in the clouds of the sky but not a bird deformation that will make the inverted image look unrealistic. Therefore, our saliency map should allocate more weight in the bird than in it allocates in the background sky.

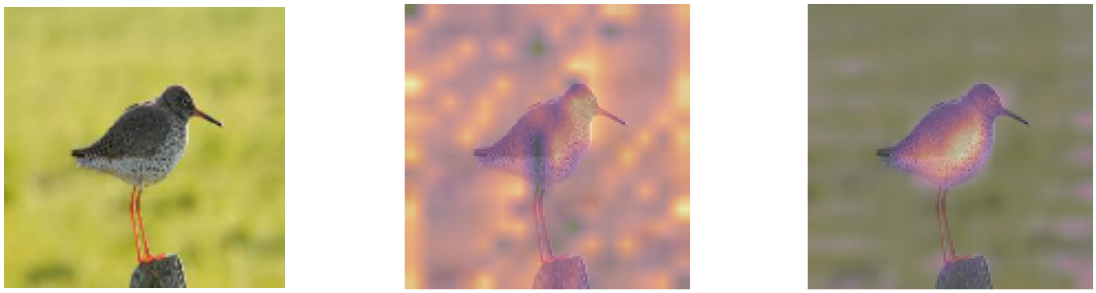
In the paper, we show that different heads specialize in discovering important image parts (for example, some heads learn to focus on local neighbhoords, important shape edges, background, etc.) so extracting a saliency map S by averaging all heads usually leads in a uniform distribution over pixels, which is not helping inversion. Figure 1b shows the saliency map jointly all heads of the attention layer of the discriminator produce. Although the bird receives a bit more attention than the background, it is not clear how this map would help weight our loss for inversion. However, as illustrated in 1c, there are heads that produce far more meaningful saliency maps for a good-looking inversion. There is a drawback here as well though; if we use that head only, we completely miss the background.

To address this problem, we find two solutions that work quite well.

- Solution 1: calculate Equation 1 separately for each head and then add the losses. In that case, the new loss function is given by the following equation:

$$\sum_i \|\big(D^0(G(z)) - D^0(x)\big) \cdot S'_i\|^2, \tag{2}$$

  where S'_i is the saliency map extracted from head i.



(a) (b) (c)

Figure 1: (a) Real image of a redshank. (b) Saliency map extracted from **all** heads of the Discriminator. (c) Saliency map extracted from a **single** head of the Discriminator. Weighting our loss function with (b) does not have a huge impact, as the attention weights are almost uniform. Saliency map from (c) is more likely to help correct inversion of the bird. We can use saliency maps from other heads to invert the background as well.

- Solution 2: Examine manually the saliency maps for each head and remove the heads that are attending mainly to non-crucial for the inversion areas, such as homogeneous backgrounds.

1.2. More inversion visualizations

We present several inversions for different categories of real images at Figure 2. In all our Figures, we use Solution 1 as it has the advantage that it does not require human supervision.

With our method, we can effectively invert real world scenes. We tested the standard inversion method [2] for these images as well and the results were far less impressive for all images. Especially for the dogs, we noted complete failure of the previous approach, similar to what we illustrate in Figure 5.

1.3. Experiments setup

In this subsection, we will briefly describe the experimental setup for our inversion technique. We choose to use the recently introduced Lookahead [9] optimizer as we find that it reduces the number of different seeds we have to try for a successful inversion. For the vast majority of the examined real images, we are able to get a satisfying inversion by trying at most 4 different seeds. We set the learning rate to 0.05 and we update for maximum 1500 steps. On a single V100 GPU, a single image inversion takes less than a minute to complete. We choose to invert real-world images that were not present in the training set. We initialize our latent variables from a truncated normal distribution, as explained in 2.

2. Truncation and how it helps inversion

In the BigGAN [3] paper, the authors observed that latent variables sampled from a truncated normal distribution generated generally more photo-realistic images compared to ones generated from the normal distribution which was used during the training. This so-called truncation trick (resampling the values with magnitude above a chosen threshold) leads to improvement in sample quality at the cost of reduction in sample variety. For the generated images of YLG presented in both the original paper and this supplementary material, we also utilized this trick.

Interestingly, the truncation trick can help inversion as well under some conditions. If the original image has good quality, then according to the truncation trick, it is more probable to be generated by a latent variable sampled from a truncated normal (where values which fall outside a range are resampled to fall inside that range) than the standard normal distribution $N(0, I)$. For that reason, in our inversions we start our trainable latent variable from a sample of the truncated normal distribution. We found experimentally that setting the truncation threshold to two standard deviations from the median (in our case 0), is a good trade-off between producing photo-realistic images and having enough diversity to invert an arbitrary real world image.

3. Strided Pattern

In the ablation studies of our paper, we train a model we name YLG - Strided. For this model, we report better results than the baseline SAGAN [8] model and slightly worse results than the proposed YLG model. The purpose of this section is to give more information on how YLG and YLG - Strided differ.

First of all, the only difference between YLG and YLG Strided is the choosing of attention masks for the attention heads: both models implement 2-step attention patterns with Full Information and two-dimensional locality using the ESA framework.

YLG model uses the RTL and LTR patterns introduced

Figure 2: More inversions using our technique. To the left we present real images and to the right our inversions using YLG SAGAN.

in the paper (see Figure 2b, 2c of the paper). Each pattern corresponds to a two-step attention: in our implementation of multi-step attention we compute steps in parallel using multiple heads, so in total we need 8 attention heads for YLG. In YLG - Strided instead of using different patterns (RTL and LTR), we stick with using a single attention pattern. Our motivation is to: (i) investigate whether using multiple attention patterns simultaneously affects performance, (ii) discover whether the performance differences between one-dimensional sparse patterns reported in the literature remain when the patterns are rendered to be aware of two-dimensional geometry. To explore (i), (ii) a natural choice was to work with the Strided pattern proposed in Sparse Transformers [4] as it was found to be (i) effective for modeling images and (ii) more suitable than the Fixed pattern (see Figure 2a), on which we built to invent LTR, RTL.

We illustrate the Strided pattern, as proposed in Sparse Transformers [4], in Figures 3a, 3c. For a fair comparison with LTR, RTL we need to expand Strided pattern in order for it to have Full Information. Figures 3b, 3d illustrate this expansion. The pattern illustrated in this Figure is ex-

actly the pattern that YLG - Strided uses. Note that this pattern attends to the same order of positions, $O(n\sqrt{n})$, as LTR and RTL. For one to one comparison with YLG, YLG - Strided has also 8 heads: the Full Information pattern is implemented 4 times, as we need 2 heads for a 2-step pattern. As already mentioned, we also use ESA framework for YLG - Strided.

4. Multiple steps and multiple heads

In the paper, we use different attention heads to implement the different steps of $2-$step attention sparsifications. However, it might still be unclear why and how attention heads are related to different attention steps, in the sense that each node in the Information Flow Graph attends to any other node in the graph. In this section, we will clarify any vague points around this matter.

The relation between multiple heads and multiple steps is indeed a confusing issue in the literature. We follow multi-stage attention exactly as implemented in [4] sec. 4.2 and in their source code. There are three ways that one could implement 2-step attention: (i) stacking two attention layers, each one implementing a stage, (ii) using a single attention layer to implement both stages or (iii) using multiple heads and combining their outputs at the end. Method (i) doubles the attention parameters and did not work as well empirically. Method (ii) introduces undesirable weight sharing: we have to multiply the first stage output again with the same matrices W_Q, W_K, W_V for the second stage, this gave poor experimental performance. Method (iii) splits stages into different heads. In this case, the stages are computed in parallel and independently. We emphasize that each head *independently* assigns attention weights to the allowed positions. In the language of Information Flow Graphs, for our $2-$step patterns this means that the attention scores between the last two vertex sets are independent with attention scores between the first two vertex sets. However, the values of different stages are combined at the end by merging the head dimension of the value tensor. This is how Full Information is maintained in the implementation: when we multiply with the values matrix, our product involves the information obtained by *all* heads.

The procedure described above is better illustrated in Figure 4. Sub-figure 4a shows the Information Flow Graph for the RTL pattern for a sequence of length 4. Sub-figure 4b shows how this $2-$step attention patter is implemented in practice. Separate heads implement the attention between subsequent vertex sets of the original Information Flow Graph independently. Then, the partial outputs of the heads are concatenated along the feature dimension. Each node in this scheme captures Full Information since its' final vector representation contains information from all other nodes in the graph.

One drawback of using multiple heads to implement multiple steps is the independence between the scores of each head in a single pass. However, we notice that as the training proceeds heads learn to co-operate (through back-propagation). Additionally, this method has the advantage that steps are computed in parallel and thus there are no performance bottlenecks.

As we noted in the paper, we also get better results using multiple attention patterns (LTR, RTL) simultaneously. We need two heads (one for each step) for each pattern and use each pattern twice. Thus, for YLG we use 8 heads total. In summary, different heads implement different stages. Masks constrain the positions that each head can attend. Multiple patterns encourage diversity and improve performance.

5. Things that did not work

In this section, we present several ideas, relevant to the paper, that we experimented on and found that their results were not satisfying. Our motivation is to inform the research community about the observed shortcomings of these approaches so that other researchers can re-formulate them, reject them or compare their findings with ours.

5.1. Weighted inversion at the generator space

We already discussed that our key idea for the inversion: we pass a real image to the discriminator, extract the attention map, convert the attention map to a saliency distribution S and we perform gradient descent to minimize the discriminator embedding distance, weighted by this saliency map:

$$\|\big(D^0(G(z)) - D^0(x)\big) \cdot S'\|^2,$$

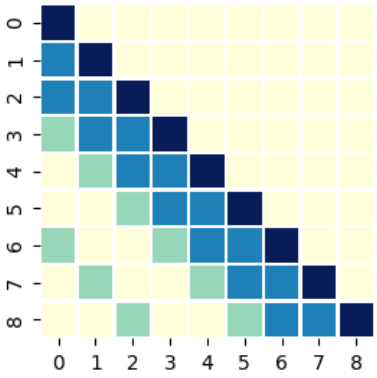
where S' is a projected version of saliency map S, x is the image, and D^0 is the Discriminator network up to, but not including, the attention layer. In practice, we use Equation 2 for the reasons we explained in Section 1 but for the rest of this Section we will overlook this detail as it is not important for our point.

Equation 1 implies that the inversion takes place in the embedding space of the Discriminator. However, naturally one might wonder if we could use the saliency map S to weight the inversion of the Generator, in other words, if we could perform gradient descent on:

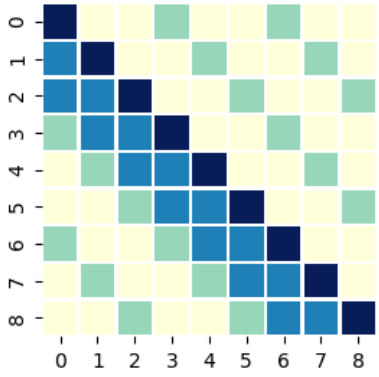
$$\|(G(z) - x) \cdot S''\|^2, \tag{3}$$

where S'' is a projected version of S to match the dimensions of the Generator network.

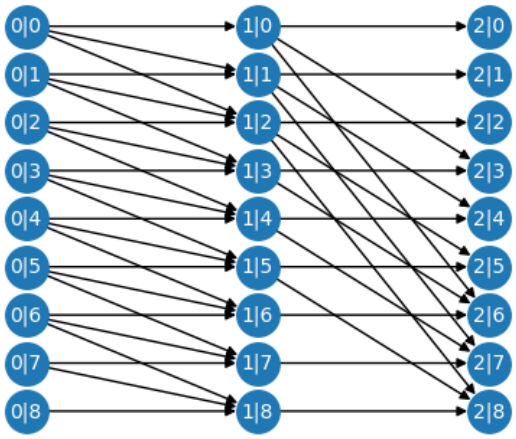
In our experiments, we find that this approach generally leads to inversions of poor quality. To illustrate this, we present inversions of an image of a real husky from the the weighted generator inversion, the weighted discriminator inversion and standard inversion method [2] at Figure 5.



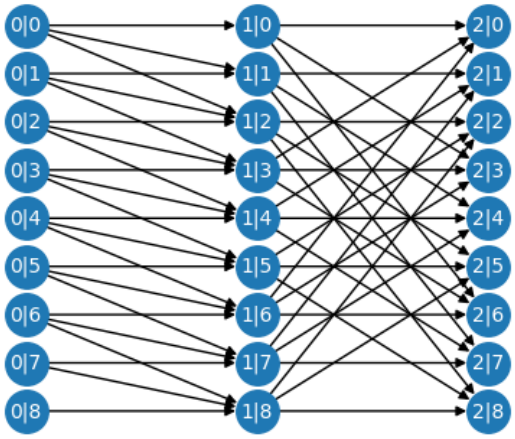
(a) Attention masks for Strided Pattern [4].



(b) Attention masks for YLG - Strided (Extended Strided with Full Information property)



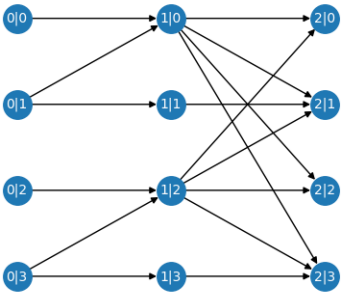
(c) Information Flow Graph associated with Strided Pattern. This pattern *does not have Full Information*, i.e. there are dependencies between nodes that the attention layer cannot model. For example, there is no path from node 2 of V^0 to node 1 of V^2.



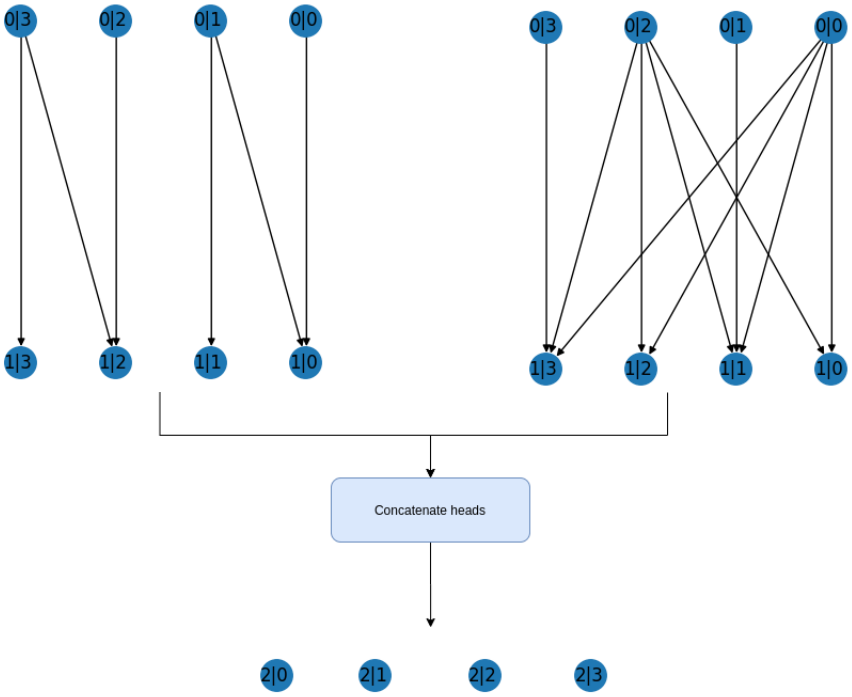
(d) Information Flow Graph associated with YLG - Strided pattern. This pattern has **Full Information**, i.e. there is a path between any node of V^0 and any node of V^2. Note that the number of edges is only increased by a constant compared to the Strided Attention Pattern [4], illustrated in 3a.

Figure 3: This Figure illustrates the original Strided Pattern [4] and the YLG - Strided pattern which has Full Information. First row demonstrates the different boolean masks that we apply to each of the two steps. Color of cell [i. j] indicates whether node i can attend to node j. With dark blue we indicate the attended positions in both steps. With light blue the positions of the first mask and with green the positions of the second mask. The yellow cells correspond to positions that we do not attend to any step (sparsity). The second row illustrates Information Flow Graph associated with the aforementioned attention masks. An Information Flow Graph visualizes how information "flows" in the attention layer. Intuitively, it visualizes how our model can use the 2-step factorization to find dependencies between image pixels. At each multipartite graph, the nodes of the first vertex set correspond to the image pixels, just before the attention. An edge from a node of the first vertex set, V^0, to a node of the second vertex set, V^1, means that the node of V^0 can attend to node of V^1 at the first attention step. Edges between V^1, V^2 illustrate the second attention step.

There are several reasons that could explain the quality gap when we change from inversion to the space of the Discriminator to that of the Generator. First of all, the saliency map we use to weight our loss is extracted from the Discriminator, which means that the weights reflect what the Discriminator network considers important at that stage. Therefore, it is reasonable to expect that this saliency map would be more accurate to describe what is important for the input of the attention of the discriminator than to the output of the Generator. Also note that due to the layers of the Discriminator before the attention, the images of the output of the generator and the input of the attention of the Discriminator can be quite different. Finally, the Discriminator may provide an "easier" embedding space for inver-



(a) RTL Information Flow Graph for $N = 4$.



(b) Implementation of RTL attention pattern ($N = 4$) in practice using separate heads for separate steps. Full Information is maintained since the final representation of each node contains information from all other nodes in the graph.

Figure 4: Implementation of multiple steps attention with multiple heads

(a) Real image.

(b) Inversion with our method.

(c) Weighted inversion at Generator.

(d) Inversion using the standard method [2].

Figure 5: Inversion with different methods of the real image of 5a. Our method, 5b, is the only successful inversion. The inversion using the weights from the saliency map to the output of the Generator, 5c, fails badly. The same holds for inversion using the standard method in the literature [2], as shown in 5d.

sion. The idea of using a different embedding space than the output of the Generator it is not new; activations from VGG16 [7] have also been used for inversion [1]. Our novelty is that we use the Discriminator instead of another pre-trained model to work on a new embedding space.

5.2. Inversion at the Discriminator space without weights

Our experimental evidence showed that weighted inversion in the Discriminator space was particularly effective. Thus, it is natural to wonder whether this inversion was successful because of the weights or just because inversion itself is easier in the Discriminator space. Although, for some images inversion on the Discriminator space gave more encouraging results comparing to the standard inversion method, it often got trapped on local minima of the loss function. Weighting the loss function in the Discriminator space consistently gave qualitatively better inversions and thus we did not expand our experiments on the idea of unweighted inversion.

5.3. Combination of dense and sparse heads

In our paper, we provide strong experimental evidence that multi-step two-dimensional sparse local heads can be more efficient than the conventional dense attention layer. We justify this evidence theoretically by modelling the multi-step attention with Information Flow Graphs and indicating the implications of Full Information. Naturally, one might wonder what would happen if we combine YLG attention with dense attention. To answer this question, we split heads into two groups, the local - sparse heads and the dense ones. Specifically, we use 4 heads that implement the RTL, LTR patterns (see paper for more details) and 4 dense heads and we train this variation of SAGAN. We use the same setup as with our other experiments. We report FID 19.21 and Inception: 51.23. These scores are far behind than the scores of YLG and thus we did not see any benefit continuing the research in this direction.

5.4. Different resolution heads

One idea we believed it would be interesting was to train SAGAN with a multi-headed dense attention layer of different resolution heads. In simple words, that means that in this attention layer some heads have a wider vector representation than others. Our motivation was that the different resolutions could have helped enforcing locality in a different way; we expected the heads with the narrow hidden representations to learn to attend only locally and the wider heads to be able to recover long-range dependencies.

In SAGAN, the number of channels in the query vector is 32, so for an 8-head attention layer normally each head would get 4 positions. We split the 8 heads into two equal groups: the narrow and the wide heads. In our experiment, narrow heads get only 2 positions for their vector representation while wide heads get 6. After training on the same setup with our other experiments, we obtain FID 19.57 and Inception score: 50.93. These scores are slightly worse than the original SAGAN, but are far better than SAGAN with dense 8-head attention which achieved FID 20.09 and Inception 46.01, as mentioned in the ablation study.

At least in our preliminary experiments, different resolution heads were not found to help very much. Perhaps they can be combined with YLG attention but we more research would be needed in this direction.

6. Information Flow Graphs

We found that thinking about sparse attention as a network with multiple stages is helpful in visualizing how information of different tokens is attended and combined. We use Information Flow Graphs (IFGs) that were introduced in [5] for modeling how distributed storage codes preserve data. In full generality, IFGs are directed acyclic graphs with capacitated directed edges. Each storage node is represented with two copies of a vertex (x_{in} and x_{out}) connected by a directed edge with capacity equal to the amount of information that can be stored into that node. The key insight is that a multi-stage attention network can be considered a storage network since intermediate tokens are representing combinations of tokens at the previous stage. The IFGs we use in this paper are a special case: every token of every stage of an attention layer is represented by a storage node. Since all the tokens have the same size, we can eliminate vertex splitting and compactly represent each storage node by a single vertex, as shown in Figure 3d.

Full information is a design requirement that we found to be helpful in designing attention networks. It simply means that any single input token is connected with a directed path to any output token and hence information (of entropy equal to one token representation) can flow from any one input into any one output. As we discussed in the paper, we found that previously used sparse attention patterns did not have this property and we augmented them to obtain the patterns we use. A stronger requirement would be that any *pair* of input nodes is connected to any pair of output nodes with two edge-disjoint paths. This would mean that flow of two tokens can be supported from any input to any output. Note that a fully connected network can support this for any pair or even for any set of k input-output pairs for $\forall k \leq n$.

An interesting example is the star transformer [6] where all n input tokens are connected to a single intermediate node which is then connected to all output tokens. This information flow graph has $2n$ directed edges and can indeed support full information. However, it cannot support a flow of 2 tokens for any pair, since there is a bottleneck at the intermediate node. We believe that enforcing good information flow for pairs or higher size sets improves the

design of attention networks and we plan to investigate this further in the future.

7. Generated images

We end our Supplementary material with some more generated images from our YLG SAGAN. The images are divided per category and are presented in Figures 6, 7, 8.

Figure 6: Generated images from YLG SAGAN divided by ImageNet category.

Figure 7: Generated images from YLG SAGAN divided by ImageNet category.

Figure 8: Generated images from YLG SAGAN divided by ImageNet category.

References

[1] David Bau, Jun-Yan Zhu, Jonas Wulff, William Peebles, Hendrik Strobelt, Bolei Zhou, and Antonio Torralba. Seeing What a GAN Cannot Generate. *arXiv e-prints*, page arXiv:1910.11626, Oct 2019. 7

[2] Ashish Bora, Ajil Jalal, Eric Price, and Alexandros G Dimakis. Compressed sensing using generative models. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 537–546. JMLR. org, 2017. 2, 4, 6

[3] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large Scale GAN Training for High Fidelity Natural Image Synthesis. *arXiv e-prints*, page arXiv:1809.11096, Sep 2018. 2

[4] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019. 3, 4, 5

[5] Alexandros G Dimakis, P Brighten Godfrey, Yunnan Wu, Martin J Wainwright, and Kannan Ramchandran. Network coding for distributed storage systems. *IEEE transactions on information theory*, 56(9):4539–4551, 2010. 7

[6] Qipeng Guo, Xipeng Qiu, Pengfei Liu, Yunfan Shao, Xiangyang Xue, and Zheng Zhang. Star-Transformer. *arXiv e-prints*, page arXiv:1902.09113, Feb 2019. 7

[7] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv e-prints*, page arXiv:1409.1556, Sep 2014. 7

[8] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7354–7363, Long Beach, California, USA, 09–15 Jun 2019. PMLR. 2

[9] Michael R. Zhang, James Lucas, Geoffrey Hinton, and Jimmy Ba. Lookahead Optimizer: k steps forward, 1 step back. *arXiv e-prints*, page arXiv:1907.08610, Jul 2019. 2