SortFormer: Permutation Encoding for Transformer

Taejin Park *
NVIDIA, Santa Clara, CA, USA taejinp@nvidia.com

Abstract

We propose *SortFormer* architecture, which is a Transformer variant that encodes permutation information into hidden states. We address the challenge of permutation invariance in processing sequential data using Transformer architectures. Despite their effectiveness in learning and generalizing across text, speech, and image data, Transformers struggle with inputs where permutations should be disregarded. This limitation often results in significant overfitting where the trained model hardly handles permutations on the unseen datapoints. We identify that the inherent design of the original Transformer architecture contributes to this deficiency. SortFormer operates by generating estimated labels at each layer, which are then sorted according to their arrival order. These sequentially ordered labels serve as a permutative encoding, guiding the Transformer model in recognizing the sequence order. Through ablation studies, we demonstrate that SortFormer exhibits reduced overfitting and enhanced generalization capabilities on unseen data sequences. Furthermore, we provide insights into the practical applicability of SortFormer in real-world scenarios. Our findings suggest that this innovative architecture holds significant promise for improving the robustness and versatility of Transformer models in handling permutation-sensitive sequential data.

1 Introduction

In recent years, the large language models (LLMs) [15, 18] gained significant popularity and most of the LLMs are based on the Transformer [20] architecture that is now being used for not only language modeling as language understanding [5, 17], GPT3 [2]. Other tasks such as image processing field also started leveraging Transformers such as image generation [14] and image recognition with ViT [6]. In speech recognition field, Conformer [8] and Whisper [16] successfully employed Transformer archietcture to speech-to-text (STT) tasks.

Transformer Variants for Efficiency The wide adoption of Transformer architecture also led to more efficient variants of the model such as Reformer [11], which introduces efficient memory handling in Transformers through locality-sensitive hashing; Transformer-XL [4], enhancing Transformer models for long sequence tasks with segment-level recurrence and a novel positional encoding; Performer [3], offering a scalable Transformer variant with a new attention mechanism for processing long sequences efficiently; Longformer [1], designed for longer texts with a modified attention mechanism that combines sliding window and global attention for tasks like document classification and question answering; SegFormer [21], an approach for semantic segmentation combining a hierarchical Transformer encoder with a lightweight MLP decoder; and RoFormer[19], which enhances Transformer models with a rotational position encoding scheme for improved performance in various natural language processing tasks. These variants showcase significant advancements in handling long sequences, improving memory efficiency, and extending the applicability of Transformer models in various domains, including text and image processing.

*Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies.

Preprint. Under review.

Definition 1 *Permutation Function π*
Let $n \in \mathbb{Z}_+$be a positive integer. Then, mathematically, we define a permutation as any invertible bijective transformation of the finite set $\{1, \ldots, n\}$ into itself. Thus, a permutation is a function $\pi : \{1, 2, \ldots, n\} \longrightarrow \{1, 2, \ldots, n\}$ such that, for every integer $i \in \{1, \ldots, n\}$, there exists exactly one integer $j \in \{1, \ldots, n\}$ for which

$$\pi(j) = i. \tag{1}$$

Definition 2 *Order Permutation Function τ*
Given a length M tuple $\mathbf{x} = (x_1, ..., x_M)$, a permutation function π is a bijection from the set of indices $\mathcal{S} = \{1, ..., M\}$ to itself. For a tuple input, the action of order permutation function τ on the original tuple in a reordered set $\mathbf{x}_\tau$ as follows:

$$\mathbf{x}_\tau = \tau\big(\mathbf{x}\big) = \tau\big(x_1, ..., x_M\big) = (x_{\pi(1)}, ..., x_{\pi(M)}) \tag{2}$$

where each element x_i is relocated to the position $\pi(i)$.

Definition 3 *Mapping Permutation Function σ*
Given a set of labeled elements $\mathcal{S} = \{x_1, x_2, ..., x_N\}$, a permutation function π is a bijection from this set onto itself, such that $\pi(x_n) = x_m$, where each element x_n is mapped to the position of another element x_m, where $n \neq m$. The function takes a tuple of elements that include repetition of elements, (e.g., $(x_1, x_2, x_2, x_3, x_3, x_1, ..., x_1,)$), and maps each element to a unique element in the same set. For a tuple input $\mathbf{x}$, the action of σ on an element in the set results in a reordered set as follows:

$$\mathbf{x}_\sigma = \sigma(\mathbf{x}) = \sigma\big(x_1, x_2, ..., x_N\big) = (\pi(x_1), \pi(x_2), ..., \pi(x_N)) \tag{3}$$

This ensures that every element x_n in the original set is uniquely associated with one and only one element x_m in the reordered set, and vice versa.

Property 1 (*Permutation Invariance*) Let $\mathcal{X}$ be a set and $\mathcal{Y}$ be a codomain. A function $f : 2^{\mathcal{X}} \to \mathcal{Y}$ is said to be *permutation invariant* if, for any subset $\{x_1, ..., x_M\} \subseteq \mathcal{X}$ and any permutation π of the indices $\{1, ..., M\}$, the following holds:

$$f\left(x_1, ..., x_M\right) = f\left(x_{\pi(1)}, ..., x_{\pi(M)}\right). \tag{4}$$

This property means that the output of f is independent of the order of the elements in its input set.

Property 2 (*Permutation Equivariance*) Let π be a permutation of $\{1, 2, ..., n\}$, and let $f : \mathbb{R}^n \to \mathbb{R}^m$ be a function. Then, f is said to be *permutation equivariant* if for every input $\mathbf{x} = (x_1, x_2, ..., x_n) \in \mathbb{R}^n$, it holds that

$$f(\tau(\mathbf{x})) = \tau(f(\mathbf{x})), \tag{5}$$

where $\tau(\mathbf{x})$ represents the permutation of the components of $\mathbf{x}$ according to τ, and $\tau(f(\mathbf{x}))$ represents the permutation of the components of $f(\mathbf{x})$ in the same manner. Thus, equivalently, the following equality holds:

$$(x_{\pi(1)}, x_{\pi(2)}, ..., x_{\pi(n)}) = (f(x_{\pi(1)}), f(x_{\pi(2)}), ..., f(x_{\pi(n)})). \tag{6}$$

Property 3 (*Permutation Order-Equivariance Mapping-Invariance*) Let X be a set, and let X^n denote the set of all n-tuples where each element belongs to X. Define a function $f : X^n \to X^n$, where f maps an n-tuple in X^n to another n-tuple in the same set. The function f is said to possess *Permutation Order-Equivariance Mapping-Invariance* if and only if for any n-tuple $\mathbf{x} = (x_1, x_2, ..., x_n)$ in X^n and any permutation of $\mathbf{x}$, denoted as $\mathbf{x}_\sigma = (\pi(x_1), \pi(x_2), ..., \pi(x_n))$, created by any bijective permutation mapping function $\sigma : X \to X$, the following equality holds:

$$f(\tau(\mathbf{x})) = \tau(f(\mathbf{x})) = \tau(f(\mathbf{x}_\sigma)) = f(\tau(\mathbf{x}_\sigma)) \tag{7}$$

where $\tau(\mathbf{x})$ and $\tau(\mathbf{x}_\sigma)$ represents the permutation of the components of $\mathbf{x}$ and $\mathbf{x}_\sigma$, respectively, as in Property 1 according to τ. Similarly, $\tau(f(\mathbf{x}))$ and $\tau(f(\mathbf{x}_\sigma))$ represent the permutation of the components of $f(\mathbf{x})$ and $f(\mathbf{x}_\sigma)$, respectively, in the same manner.

Example 1 (*Permutation Order-Equivariance Mapping-Invariance*) From a set of n-tuples, X^n from a set $X = \{a, b, c\}$, $\mathbf{a} = (a, a, b, b, c)$. If *Permutation Order-Equivariance Mapping-Invariance* holds for a function f, then the following holds:

$$f(\mathbf{a}) = f(a, a, c, c, b) = f(b, b, a, a, c) = f(b, b, c, c, a) = f(c, c, a, a, b) = f(c, c, b, b, a). \quad (8)$$

Permutation Invariance/Equivariance for Multi-head Attention The multi-head attention (MHA) architecture proposed in [20] is a key component of the Transformer model, which can be described as follows:

$$\text{MHA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\mathbf{O}_1, ..., \mathbf{O}_h)\mathbf{W}^O, \quad (9)$$

where $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{n \times d}$ are the query, key, and value matrices, respectively, and $\mathbf{W}^O \in \mathbb{R}^{hd \times d}$ is a trainable parameter matrix. And each head is defined as:

$$\mathbf{O}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V), \quad (10)$$

where $\mathbf{W}_i^Q, \mathbf{W}_i^K, \mathbf{W}_i^V \in \mathbb{R}^{d \times d/h}$ are trainable parameter matrices.

Problem that SortFormer solves We propose a model designed for the simultaneous estimation of class presences from a sequence of input tokens. Consider a set of frame-wise F-dimensional embedding vectors $\{\mathbf{x}_t\}_{t=1}^T$, where $\mathbf{x}_t \in \mathbb{R}^D, t = 1, 2, ..., T$, representing the frame index. Given the input sequence, the model is expected to generate the estimated sequence $\{\mathbf{y}_t\}_{t=1}^T$, where $\mathbf{y}_t \in \mathbb{R}^K, t = 1, 2, ..., T$. SortFormer is tasked with estimating the class presences $\{\mathbf{y}_t\}_{t=1}^T$. In this context, $\mathbf{y}_t = [y_{1,t}, y_{2,t}, ..., y_{K,t}]^\top$ denotes the class presences of K classes at time t. SortFormer operates under the assumption that $y_{k,t}$ is conditionally independent given the embedding vectors (features). Therefore, Sortformer is employing *Sigmoid* instead of *Softmax* unlike the activation function for the output layer in [20, 15]. This assumption is formalized as:

$$P\left(\mathbf{y}_1, ..., \mathbf{y}_T \mid \mathbf{x}_1, ..., \mathbf{x}_T\right) = \prod_{k=1}^{K} \prod_{t=1}^{T} P\left(y_{k,t} \mid \mathbf{x}_1, ..., \mathbf{x}_T\right). \quad (11)$$

Under this framework, the task at hand is construed as a multi-label classification problem, which is amenable to modeling via a neural network, denoted as f_Θ. The model is defined as:

$$(\mathbf{p}_1, ..., \mathbf{p}_T) = f_\Theta\left(\mathbf{x}_1, ..., \mathbf{x}_T\right), \quad (12)$$

where $\mathbf{p}_t = [p_{1,t}, ..., p_{K,t}]^\top \in (0, 1)^S$ represents the posterior probabilities of the presences of S classes at frame index t and f represents the Sortformer model with parameter. Each $y_{k,t}$ is defined as follows depending on the value of $p_{k,t}$:

$$y_{k,t} = \begin{cases} 1 & \text{if } p_{k,t} > 0.5 \\ 0 & \text{if } p_{k,t} \leq 0.5 \end{cases} \quad (13)$$

The estimation of class presences $\{\hat{\mathbf{y}}_t\}_{t=1}^T$ is given by:

$$(\hat{\mathbf{y}}_1, ..., \hat{\mathbf{y}}_T) = \underset{\mathbf{y}_t \in \mathbf{Y}}{\arg\max} \big\{ P\left(\mathbf{y}_1, ..., \mathbf{y}_T \mid \mathbf{x}_1, ..., \mathbf{x}_T\right) \big\}, \quad (14)$$

The equation 14 can be rewritten in matrix form by concatenating the vectors $\mathbf{X} = [\mathbf{x}_1 \mathbf{x}_2 ..., \mathbf{x}_T]$, where each column of $\mathbf{Y}$ represents the class presences at time t, $\mathbf{Y} = [\mathbf{y}_1\, \mathbf{y}_2\, ...\, \mathbf{y}_T]$. In the simplest form, the model can be represented as:

$$\mathbf{Y} = f_\Theta\left(\mathbf{X}\right), \quad (15)$$

where $\mathbf{X} \in \mathbb{R}^{D \times T}$ is the input sequence and $\mathbf{Y} \in \mathbb{R}^{K \times T}$ is a binary matrix contains an output sequence.

Task Description In this problem, the core challenge lies in the output sequence $\mathbf{Y}$, where the number of frames have non-zero $\mathbf{y}_t$ is determined by the model's estimation based on the input sequence $\mathbf{X}$. Here, $\mathbf{X}$ represents an unseen type of data. The model's task is to count the number of distinct classes within the input sequence and generate corresponding class labels for each identified class.

It is crucial to make a clearn distinction between the problem that SortFormer is tackling and the well known tasks such as clustering, identification tasks. This problem shares similarities with clustering tasks, open-set recognition tasks where the objective is to count the number of clusters and assign labels to them. However, there are follwing key characteristics that distinguish this problem from the aforementioned tasks:

- **Infer Unseen Data Class** The task is not a typical classification task, where the model needs to classify the unseen type of class. For example, in open-set face indexing, the model needs to identify the face of people that have not been seen during training time.
- **Multi-labels** Each token into SortFormer can have multiple labels. For example, one picture can contain multiple people's faces. Most clustering systems are not designed to assign multiple labels to a single data point.
- **Non-Event Classification** The system should detect "class absence", which requires the model to learn the characteristics of null-class characteristics of the feature input. Clustering methods typically do not accommodate cases with 'void class', where no discernible patterns exist within a given frame.
- **Set-based Label Assignment Loss** The model should count the number of sets and label each frame accordingly to get generic identifiers. Followingly, the loss should be calculated based on the set-based label assignment. This aspect is often handled with permutation invariant loss (PIL).
- **Small-scale Set Count** The task is expecting $K < 10$ classes, which is a small-scale set count. Most of the neural network based clustering systems are designed to handle large-scale set count where $K > 1000$. This also decreases the burden of applying sorting algorithm on the output classes.

An illustrative example is the speaker diarization problem. In speaker diarization, the input sequence is an audio signal, and the output sequence comprises generic identifiers (e.g., speaker1, speaker2, ..., speakerN) for each frame. However, challenges arise in frames where no speaker is present, resulting in an output $\mathbf{y}_t$ that consists entirely of zeros. Additionally, complications occur in frames with multiple speakers, where $\mathbf{y}_t$ becomes a binary vector containing multiple ones.

This framework is not only relevant to the speaker diarization task but is also applicable to a variety of other problems, such as anomaly detection in time series data, open-set recognition in image processing, open-set facial entity indexation in video streams, unknown language detection in natural language processing, and new genre detection in music classification. the following table summarizes the tasks suitable for SortFormer and their corresponding representations.

Set-based label assignment loss has been widely investigated in audio signal processing field for speaker diarization tasks and source separation tasks.

Contributions The proposed method has following contributions:

- **Replace Permutation Invariant Loss** We show that sorting layers and skip connections can be used to replace permutation invariant loss (PIL) for the tasks that require permutation invariance treatment. SortFormer can be trained by regular cross-entropy loss without any permutation-free mechanism.
- **From Passive to Active Permutation Resolution** We define the methods that apply permutation invariant treatment at the loss calculation layer as passive permutation resolution. We define the methods that apply permutation invariant treatment at the encoder model architecture level as active permutation resolution. The active permutation resolution has not only the advantage of reducing the computational complexity during training time but also the advantage of handling long-context or streaming settings.
- **Improved Compatibility for Multi-modal Training** We show that the proposed architecture is designed to be trained with the other embeddings or modalities to be trained with a single

Task	Feature Representation	Input Data Type	Output Description
Speaker Diarization	Speaker embeddings	Audio streams	Identifiers for each speaker; 'no label' for silent frames
Anomaly Detection	Signal feature embeddings	Time series data	Anomaly identifiers; 'no label' for normal data points
Open-set Face Indexation	Face embeddings	Video streams	Anonymized identifiers for each face; 'no label' for non-face frames
Unknown Language Detection	Text embeddings	Text data	Language identifiers; 'no label' for unrecognizable languages
New Genre Detection	Music feature embeddings	Audio tracks	Music genre identifiers; 'no label' for unconventional genres

Table 1: Data Analysis Tasks and their Representations

Transformer-vairant model on top of SortFormer layer. To achieve this, Transformer encoder type of architecture is the best fit for training the entire model with a single loss and label type.

2 Prior Works

Clustering There have been numerous studies on making Transformer perceive longer context, model compressing for memory efficiency and sparse models. However, there have been limited number of published work regarding general-purpose transformer variants which can handle permutation

Deep clustering [9] approach uses a permutation invariant objective. The permutation invariant objective is executed by employing a K-dimensional embedding $V = f_\theta(x) \in \mathbb{R}^{N \times K}$, parameterized by θ, such that performing some simple clustering in the embedding space will likely lead to a partition of $\{1, ..., N\}$ that is close to the target one. In this work, $V = f_\theta(x)$ is based on a deep The partition-based training requires a reference label indicator $Y = \{y_{n,c}\}$, mapping each element n to each of c arbitrary partition classes, so that $y_{n,c} = 1$ if element n is in partition c.

$$C(\theta) = \left|VV^T - YY^T\right|_W^2 \tag{16}$$

This objective has been used widely, this objective has not been applied to Transformer architecture. Moreover, clustering loss is mean sequare error loss which is different from the loss function for other tasks (e.g., cross entropy) to be trained with Transformer architecture. In this paper, we categorize permutation-invariant loss calculation as passive permutation resolution approach, since the model itself does not have any mechanism to handle permutation. Thus, during inference time, no permutation handling is done.

Permutation Invariant Neural Networks As opposed to passive permutation resolution, there have been a few studies on making neural networks to be permutation invariant, which can be categorized as active permutation resolution. In active permutation resolution approaches, the model itself has a mechanism to handle permutation while making it permutation invariant or equivariant and these permutation handling mechanism happens during both training and inference time. The most well known approach is Deep sets [23]. In [23] the author proposed a permutation invariant network where a function $f(X)$ operating on a set X having elements from a countable universe, is valid set function, i.e., invariant to the permutation of instances in X, iff it can be decomposed in

the form $\rho\left(\sum_{x \in X} \phi(x)\right)$, for suitable transformations ϕ and ρ. The proposed methods in Deep Sets can either handle permutation-order invariant or equivariant. As we discssed in the previous section, the task that SortFormer is tackling is Permutation Order-Equivariance Mapping-Invariance, not permutation order invariant. More importantly, if we exclude positional encoding, the Transformer architecture is permutation order equivariant.

This led to permutation order invariant architecture Set Transformer [13], which employed pooling by multi-head attention (PMA) to replace the sum or mean pooling operation for $\phi(x)$ functions in Deep Sets [23]. The author in [13] compared the performance of the regular pooling methods and PMA and showed that PMA outperforms the regular mean, max or sum pooling methods. The Set Transformer is closely related to SortFormer in the sense that both models perform clustering task. However, Set Transformer [13] is designed to make Transformer architecture permutation order invariant, while SortFormer is designed to make Permutation Order-Equivariant Mapping-Invariant. In addition, Set Transformer requires additional hyper parameters such as inducing points m and seed vector count k. Another difference is that Set Transformer is not designed to handle the non-event class nor the overlap of two classes where *Softmax* is used for the output layer. Although the problem Set Transformer is tackling is not exactly the same as SortFormer, the model can be used as a baseline model for comparison with SortFormer.

Permutation Invariant Loss There is another set of studies that have exactly the same objective as SortFormer with different approaches. The concept of permutation invariant loss (PIL) or permutation invariant training (PIT) was first popularized by the two studies [12, 22] for the task of speech source separation tasks which inevitably require the model to handle permutation invariant loss calculation. Followingly, [7] adopted the concept of PIL for the task of speaker diarization, and later improved in [10] by employing a sequence-to-sequence model to generate the attractors by training the model with PIL. While PIL shows promising results in the aforementioned tasks, it has a few limitations compared to the sorting-based mechanism.

First, PIL requires O(N!) time complexity with a brute-force method or at least O(N^3) time complexity for N number of classes to calculate the loss with the linear sum assignment algorithm that finds the best permutation mapping between the estimated labels and ground truth labels. This becomes a huge burden when the number of classes is large. The sorting-based loss calculation can be done in O($Nlog(N)$) time with numerous sorting algorithms and even can be done in O(N) time if the sorting is based on counting sort algorithm since the keys are binary numbers.

Second, we need a special type of loss function at the output layer of the model. This becomes a limitation when we want to train a model with a few different tasks at the same time with one ground truth. For example, we want to train a model with speech recognition, translation and speaker diarization tasks at the same time. The sorting-based mechanism does not demand any special loss function at the output layer; it can be trained with any loss function once the ground truth labels are sorted.

Lastly, PIL-based models have a chance to learn the correlation between the estimated labels and the input features. This is because the model does not have any information regarding permutation until it generates the estimated labels at the output layer. The sorting-based mechanism alleviates the problem since the model is informed by the permutation encoding at each layer. Thus, when it comes to streaming setup or long-context settings, PIL-trained models need to match the permutation with the old and new estimated labels, while the sorting-based mechanism can simply assign the labels in terms of arrival orders.

3 Multi-speaker ASR Model

3.1 Speaker Encoding

We employ a novel way to inform the estimated speaker labels to the model by encoding the speaker labels into the decoder part of the model. We propose two different setups.

3.1.1 Additive Kernel-based Speaker Encoding

The first setup is kernel-based speaker encoding, where the speaker labels are encoded by an additive kernel function. The following equation is the kernel-based speaker encoding:

$$\mathbf{Y} = \|\mathbf{X}\|_2 + \mathbf{K}^T \cdot \mathbf{S} \tag{17}$$

where $\mathbf{X}$ is D by T ASR encoder embedding, $\mathbf{K}$ is a N_s by D kernel matrix, $\mathbf{S}$ is a N_s by T speaker label matrix, and $\mathbf{Y}$ is the final speaker-encoded input embedding to the decoder, which has also D by T.

3.1.2 Angular Speaker Encoding

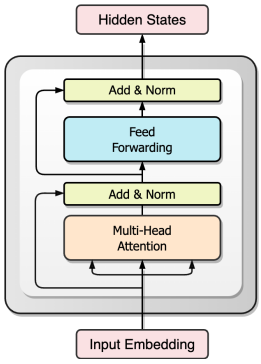

Figure 1: Transformer Encoder

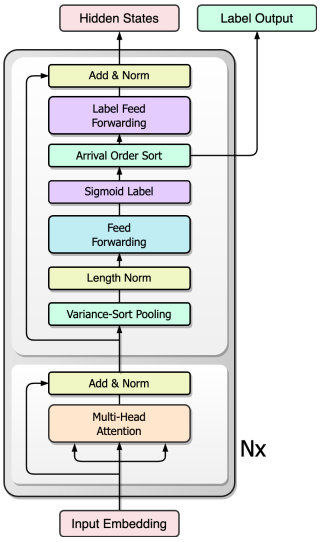

Figure 2: SortFormer Encoder

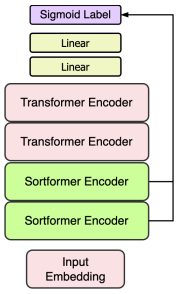

Figure 3: SortFormer Example

References

[1] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[2] Tom B Brown, Benjamin Mann, Nick Ryder, Mike Subramanian, Jared Kaplan, Praveen Dhariwal, Dario Neel, Pranav Shyam, Adam Mishkin, and Alec Radford. Gpt-3: Language models are few-shot learners. *arXiv preprint arXiv:2001.08237*, 2020.

[3] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[4] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Niu, Kristina Toutanova, Yonatan M. Belinkov, Michael Peters, Vamsi Nair, and Paul Poliakoff. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Advances in neural information processing systems*, pages 9385–9404, 2018.

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[7] Yusuke Fujita, Naoyuki Kanda, Shota Horiguchi, Yawen Xue, Kenji Nagamatsu, and Shinji Watanabe. End-to-end neural speaker diarization with self-attention. In *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 296–303. IEEE, 2019.

[8] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, et al. Conformer: Convolution-augmented transformer for speech recognition. *arXiv preprint arXiv:2005.08100*, 2020.

[9] John R Hershey, Zhuo Chen, Jonathan Le Roux, and Shinji Watanabe. Deep clustering: Discriminative embeddings for segmentation and separation. In *2016 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 31–35. IEEE, 2016.

[10] Shota Horiguchi, Yusuke Fujita, Shinji Watanabe, Yawen Xue, and Paola Garcia. Encoder-decoder based attractors for end-to-end neural diarization. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 30:1493–1507, 2022.

[11] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2019.

[12] Morten Kolbæk, Dong Yu, Zheng-Hua Tan, and Jesper Jensen. Multitalker speech separation with utterance-level permutation invariant training of deep recurrent neural networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(10):1901–1913, 2017.

[13] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pages 3744–3753. PMLR, 2019.

[14] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Llion Jones, Aidan N Gómez, Illia Polosukhin, Lukasz Kaiser, and Illia Polosukhin. Image transformer. *arXiv preprint arXiv:1802.05720*, 2018.

[15] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[16] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *International Conference on Machine Learning*, pages 28492–28518. PMLR, 2023.

[17] Colin Raffel, Noam Shazeer, Adam Roberts, Kim Lee, Nikita Parekh, and Karthik Narasimhan. Learning to transfer the effectiveness of pretraining to downstream tasks. In *International Conference on Learning Representations*, 2019.

[18] Rajkrishna Rao, Chenglong Chen, Marcus Hendricks, Mathias Kreutzer, Rasa Li, Michael Mishkin, Noah Price, Shafiq Puri, Muhammed Qadir, Siva Reddi, Vikas Shobhana, Yash Talwar, Yen-Ling Tay, Tong Wang, Ming Wei, Eric Wilcox, Yu Wu, Yuan Yuan, Pengcheng Zhang, and Yu Zhou. Megatron-turing nlg: Training an extensively labeled 530b parameter language model. *arXiv preprint arXiv:2201.11991*, 2022.

[19] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, page 127063, 2023.

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[21] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. *Advances in Neural Information Processing Systems*, 34:12077–12090, 2021.

[22] Dong Yu, Morten Kolbæk, Zheng-Hua Tan, and Jesper Jensen. Permutation invariant training of deep models for speaker-independent multi-talker speech separation. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 241–245. IEEE, 2017.

[23] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.