Learning to Sort in Continuous Spaces: Neural Architectures and Iterative Training Strategies

Matthieu CORNU
Thales DMS France, Elancourt
Systèmes et Applications des Technologies de l'Information et de l'Energie, Université Paris-Saclay, ENS Paris-Saclay, CNRS, Paris-Saclay
matthieu.cornu@thalesgroup.fr

Cyrille ENDERLI
Thales DMS France, Elancourt
cyrille-jean.enderli@fr.thalesgroup.com

Thomas RODET
Systèmes et Applications des Technologies de l'Information et de l'Energie, Université Paris-Saclay, ENS Paris-Saclay, CNRS, Paris-Saclay
thomas.rodet@ens-paris-saclay.fr
ORCID: 0000-0003-0473-9390

***Abstract*—Sorting is a fundamental problem in computer science and plays a crucial role in mechanistic interpretability due to its direct link with comparison and ordering mechanisms. Recent research has explored AI-assisted sorting, primarily in discrete spaces where categorical encoding methods, such as one-hot encoding, can be used. However, in continuous domains like $\mathbb{R}$, these methods become impractical, requiring models to develop fine-grained differentiation abilities. In this work, we investigate the capacity of neural networks to learn an order relation in a continuous setting. We evaluate CNN and Transformer architectures and show that high performance can be achieved by employing an iterative learning strategy over progressively harder to sort spaces.**

***Index Terms*—artificial intelligence, sequence processing, sorting, accuracy, learning**

I. Introduction

The ability of artificial intelligence to capture similarities and differences within elements of a sequence is a fundamental challenge in many areas of information processing. Major applications include anomaly detection in time series [1], sequence alignment in bio-informatics [2], and weather forecasting [3]. This work on determining order relationships through learning is also relevant to AI-driven sensor modelling for radar signal processing. Accurately reproducing the interaction processes requires the model to correctly discern these pulses according to an understanding of their underlying interaction laws, which is crucial for identifying characteristic patterns in radar signatures. In recent years, rapid advancements in sequence processing, particularly with the rise of generative language models such as GPT-based architectures [4], have significantly broadened AI's application landscape in this field. However, when the space of elements is continuous rather than discrete—unlike the human language vocabulary—traditional encoding techniques such as one-hot encoding become very difficult to apply. This method, commonly used to represent categorical data [5], relies on a unique mapping between each element and a dimension in a vector space. In a continuous space like $\mathbb{R}$, this would require an infinite number of dimensions, making the approach impractical. As a result, the ability of models to distinguish between similar values becomes a critical challenge. To explore this issue, we focus on sorting, a key problem in mechanistic interpretability [6] due to its link with comparison and ordering mechanisms. As illustrated in Figure 1, the sorting problem consists of reordering lists of elements based on an element-wise score computed according to an implicit criterion (e.g., the first coordinate of each element). Hence, the model must understand how this criterion influences the order and correctly sort elements, even when their scores are very close.

Historically, optimizing sorting algorithms has been a central challenge in computer science, from early classical implementations (quicksort, mergesort) to modern approaches leveraging parallelism and distributed computing. Today, the use of AI to accelerate these algorithms is garnering increasing interest. Tim Kraska et al. proposed a learned database system known as SageDB [7], which predicts each element's position based on prior training on the data, followed by a traditional sorting algorithm for refinement. This idea was extended by Xiaoke Zhu et al. [8], who iteratively apply a coarse sorting step using a learned cumulative distribution function until a conflict threshold is met, followed by a traditional sorting algorithm. Another similar approach, Balanced Learned Sort by Paolo Ferragina et al. [9], classifies elements into buckets based on learned distribution patterns, allowing for classical bucket-by-bucket sorting. The common point among these methods is the use of AI as a pre-sorting tool integrated into a larger algorithm. In contrast, we aim to focus on a problem where the order relation is not explicitly defined but must instead be learned by the model only from data. Specifically, we seek to train models capable of discovering and generalizing an implicit ordering relation that emerges from vectors in $\mathbb{R}^d$, without any prior knowledge of the underlying structure.

More in line with our approach, Jungtaek Kim et al. [10] trained a transformer to determine the sorted indices of a sequence, where elements are multi-digit images. Mateusz Baginski et al. [11] and Eionar Urdshals et al. [6] similarly explored the sorting capabilities of transformers from a Mechanistic Interpretability perspective. In their studies, transformers receive an unordered list and directly predict the sorted sequence. Again, the element space consisted of integers, ranging from 1 to 64 in the first article and 1 to 4000 in the second. The latter approach is significantly different from ours, as it relies on one-hot encoding, which artificially separates each element and, by extension, each possible value of the sorting criterion.

To our knowledge, all existing work on sorting either assumes prior knowledge of the problem, integrating AI into a larger algorithm, or focuses on sorting elements within a finite space. In contrast, we aim to address sorting in continuous spaces, where one-hot encoding is not feasible, introducing the added challenge of distinguishing closely related values.

Our contributions are as follows: we define a learning problem focused on order relations to analyse neural networks' fine-grained discernment capabilities; we propose a loss function that ensures balanced sequence contributions; we evaluate CNN [12] and Transformer [13] architectures; and we show that high performance can be achieved with an iterative learning strategy on progressively expanding spaces. Our proposed solution aligns with the principles of Continual Learning and Curriculum Learning [14], which suggest that mastering a complex task is more effective through the successive learning of increasingly complex sub-tasks.

The article is structured into four parts. First, we define the sorting problem in detail and highlight the challenges of training AI for this task. Next, we describe our proposed methods that enable effective AI training and performance. Then, we introduce the tested architectures. Finally, we present our results along with the evaluation metrics.

II. The Sorting Procedure and its Challenges

The ordering relation is defined through a hyper-parameter vector V_S, which can be chosen randomly. For a sequence of vectors $V_1, V_2, \cdots, V_n$, the sorting procedure involves computing a score $S_i = \langle V_i, V_S \rangle$ for each vector, where $\langle \cdot, \cdot \rangle$ denotes the inner product. This score measures the alignment or similarity between V_i and the reference vector V_S. The list $[(V_1, S_1), (V_2, S_2), \cdots, (V_n, S_n)]$ is then sorted in ascending order based on these scores, resulting in an ordered sequence of vectors. The permutation σ that sorts the scores ensures that $S_{\sigma(1)} < S_{\sigma(2)} < \cdots < S_{\sigma(n)}$ and so the ordered sequence is $V_{\sigma(1)}, V_{\sigma(2)}, \cdots, V_{\sigma(n)}$.

In figure 1, we illustrate using a 10-element sequence the 4-step sorting procedure, which can be summarized as follows:

- Input: A sequence of vectors is provided as input

$$[V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9, V_{10}]$$

- Scoring: For each vector V_i, a score S_i is computed

$$[(V_1, S_1), (V_2, S_2), (V_3, S_3), (V_4, S_4), \cdots, (V_{10}, S_{10})]$$

- Sorting: As the scores are sorted as

$$S_3 < S_2 < S_8 < S_4 < S_1 < S_{10} < S_7 < S_6 < S_9 < S_5$$

we sort the list:

$$[(V_3, S_3), (V_2, S_2), (V_8, S_8), (V_4, S_4), \cdots, (V_5, S_5)]$$

- Output: The scores are discarded, and the final output is the ordered sequence of vectors

$$[V_3, V_2, V_8, V_4, V_1, V_{10}, V_7, V_6, V_9, V_5]$$

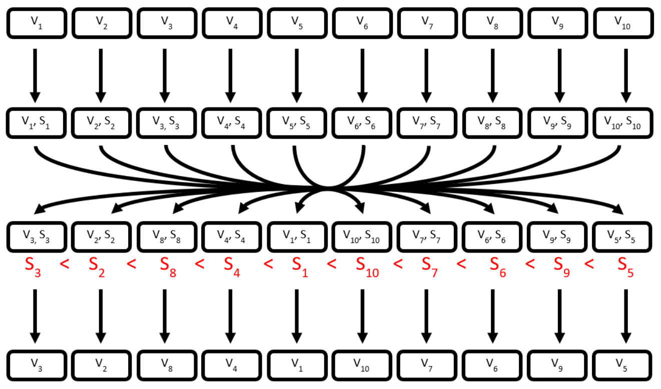
V1 V2 V3 V4 V5 V6 V7 V8 V9 V10
V1, S1 V2, S2 V3, S3 V4, S4 V5, S5 V6, S6 V7, S7 V8, S8 V9, S9 V10, S10
V3, S3 V2, S2 V8, S8 V4, S4 V1, S1 V10, S10 V7, S7 V6, S6 V9, S9 V5, S5
S3 < S2 < S8 < S4 < S1 < S10 < S7 < S6 < S9 < S5
V3 V2 V8 V4 V1 V10 V7 V6 V9 V5

Fig. 1. Diagram of the sorting mechanism considered.

The challenge for an AI learning to solve this problem lies in the wide diversity of sequences it must sort. The model must be capable of handling sequences where the scores of the elements are very close to one another, as well as sequences where the scores are widely dispersed. Since the AI has no prior knowledge of the sorting rule, it can only infer the procedure by learning from the training data. This understanding must be particularly precise and nuanced when dealing with sequences hardly separable, i.e., where the scores are close relative to the range of possible values they can take. To support these intuitions, we conducted simulations of the learning process on datasets of increasing complexity. The training is performed on windows (defined in Section III) with a scaling parameter λ ranging from 1 (simple sequences) to 10^{-4} (complex sequences).

Figure 2 illustrates the increasing difficulty faced by AI models in learning to sort sequences as data complexity grows. We compare the performance of our two architectures: Transformer and CNN. The solid curves represent the average performance across all sequences, highlighting the advantage of the Transformer over the other architectures. Performance is defined by two metrics: the error metric, which quantifies the average distance between the predicted sequences and the correctly sorted sequences (3), and the accuracy metric, which measures the fraction of elements correctly reordered by the model on average (6).

We add boxplots of the Transformer's sequence-wise performance distribution for a given λ to highlight its performance disparities. The CNN exhibits similar disparities, which are not displayed for clarity.

All architectures encounter similar difficulties when learning on sequences with lower values of λ, which, in this configuration, is linked to the ratio eq. (4), defined as $r = \lambda/20$. This suggests that the challenge is not specific to any particular architecture but rather inherent to the problem itself. Beyond

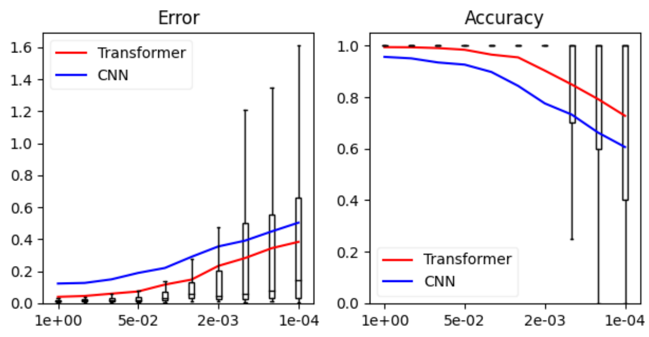

Fig. 2. Error and Prediction Accuracy vs. λ for CNN and Transformer.

$r = 10^{-4}$, we observe that the model begins to struggle with sorting certain sequences.

III. Method

In this section, we delve into the details of the proposed learning process. Specifically, we present the error metric that corresponds to the loss minimized during training eq. (3), which incorporates a unique weighting scheme, and we explain our progressive learning strategy, including the definition of the windows mentioned earlier. The error metric is based on the Euclidean norm of the difference between a sequence predicted by the model (P_k) and the ideally sorted sequence (T_k). First, for two sequences P_k and T_k, each consisting of n vectors of dimension d, we define the sequence error as:

$$\mathrm{E}(P_k, T_k) = \frac{\|P_k - T_k\|_2}{\sqrt{n \times d}} \tag{1}$$

The sequence error represents a form of distance between a predicted sequence and a sorted sequence. To provide context for this term, we also estimate the sequence error produced by the most naive approach to reducing it. For a sequence T_k, the vector P that minimizes $\mathrm{E}(\mathrm{Rep}(P), T_k)$, where $\mathrm{Rep}(P) = [P, P, \cdots, P]$ (a sequence of n repetitions of P), is the mean vector $\bar{T}_k$ of the sequence T_k. The term $\mathrm{E}_k = \mathrm{E}(\mathrm{Rep}(\bar{T}_k), T_k)$ represents the sequence error achieved simply by predicting the mean vector. As a normalisation factor, E_k ensures that, regardless of the mean amplitude of the vectors in a sequence, all sequences contribute equally to the total error : The weighted sequence error of the pair of sequences P_k and T_k is :

$$\mathrm{err}(P_k, T_k) = \frac{\mathrm{E}(P_k, T_k)}{\mathrm{E}(\mathrm{Rep}(\bar{T}_k), T_k)} \tag{2}$$

Finally, we sum over batch dimension b to get our custom loss. It computes the weighted average of distances between predicted and sorted sequences :

$$\mathrm{Err}(\mathbf{P}, \mathbf{T}) = \sqrt{\frac{1}{b}\sum_{k=1}^{b} \mathrm{err}(P_k, T_k)^2} \tag{3}$$

The concept of a window is central to our learning strategy. A window defines a range for the mean ($[\mu_{min}, \mu_{max}]$) and standard deviation ($[\sigma_{min}, \sigma_{max}]$) of the score values across sequences in a set. Specifically, a window is defined as the Cartesian product $[\mu_{min}, \mu_{max}] \times [\sigma_{min}, \sigma_{max}]$. Thus, a set is said to be characterized by the window $[\mu_{min}, \mu_{max}] \times [\sigma_{min}, \sigma_{max}]$ if its sequences are generated such that, the score distribution within a sequence is defined by parameters sampled as follows :

- The mean is uniformly drawn from $[\mu_{min}, \mu_{max}]$.
- The standard deviation is logarithmically drawn $[\sigma_{min}, \sigma_{max}]$.

This approach allows us to control the minimum precision required to sort the most challenging sequences, which can be approximated by the ratio

$$r = \sigma_{min}/(\mu_{max} - \mu_{min}) \tag{4}$$

Our training strategy follows a progressive difficulty increase. Initially, the model is trained on sequences that are easily separable, meaning their standard deviation is on the same order of magnitude as the range of score values. At this stage, the model can correctly sort sequences without fully grasping the underlying mechanism. As training progresses, we gradually introduce sequences with decreasing separability, requiring the model to refine its sorting process. This transfer learning approach enables the model to progressively master the task and successfully sort sequences of varying difficulty, from easily separable to highly challenging. The training dataset evolves through a sequence of training windows defined as:

$$[\mu_{min}, \mu_{max}] \times [\lambda, \sigma_{max}]$$

where λ controls the minimum separability of the sequences thanks to this link with the ratio eq. (4) : $\lambda = r \times (\mu_{max} - \mu_{min})$. By adjusting λ, we systematically vary the difficulty of the sequences presented to the model during training.

IV. Architectures

We compare the results obtained using two architectures renowned for their effectiveness in sequence processing: a Convolutional Neural Network (1.9M parameters) and a Transformer network (0.6M parameters). For the CNN, we adopt the architecture proposed by Kalchbrenner et al. [15] with 10 layers and 10 neurons per layer. While this architecture is effective for sequence tasks, our primary focus is on the Transformer, which has demonstrated superior performance in handling complex dependencies.

For the Transformer, we use the encoder from Vaswani et al. [13]. As shown in Figure 3, it consists of 10 layers with 4-head self-attention in a 128-dimensional space. This mechanism captures dependencies between elements while positional encodings preserve order—both essential for sorting. By weighting input features, the model focuses on relevant parts of the sequence, making it well-suited to our task

By comparing these architectures, we aim to confirm that the learning difficulty is not linked to a specific architecture but is a fundamental challenge of the sorting problem. The

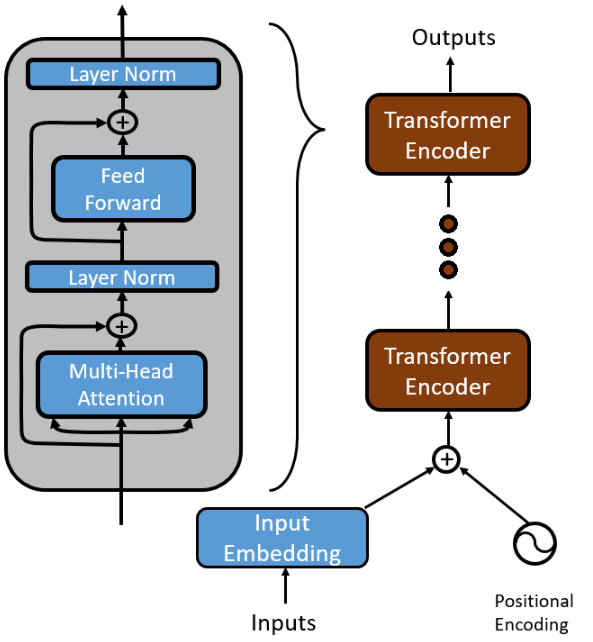

Fig. 3. Our Transformer encoder architecture has 10 layers, each featuring 4-head self-attention and a feed-forward network. Positional encodings are added to input embeddings.

Transformer, with its ability to model complex dependencies, is expected to outperform the CNN.

V. Results

To evaluate the performance of these architectures, we use two complementary metrics:

- The first metric is the training loss, referred as the error eq. (3).
- The second metric evaluates the proportion of vectors correctly repositioned by the model and is referred as the accuracy. Specifically, it measures how often the predicted sequence matches the sorted sequence in terms of vector positions. For each vector in a predicted sequence, we check if the closest vector in the sorted sequence (based on Euclidean distance) is in the same position. To formalize this, in a set of b sequences composed of n elements, for a given pair of predicted sequence $\mathbf{P}_k$ and sorted sequence $\mathbf{T}_k$, we denote their i_{th} elements as $\mathbf{P}_{k,i}$ and $\mathbf{T}_{k,i}$. The accuracy for this pair of sequences is computed as:

$$\operatorname{acc}(\mathbf{P}_k, \mathbf{T}_k) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{\{\mathbf{T}_{k,i}\}}\big(\operatorname*{argmin}_{\mathbf{T}_{k,j}\in\mathbf{T}_k}(\|\mathbf{P}_{k,i} - \mathbf{T}_{k,j}\|_2)\big) \tag{5}$$

where $\mathbb{1}$ is the indicator function. Finally, the overall accuracy over a batch of b sequences is computed as the average of the accuracy across all pairs $(\mathbf{P}_k, \mathbf{T}_k)$:

$$\operatorname{Acc}(\mathbf{P}, \mathbf{T}) = \frac{1}{b}\sum_{k=1}^{b} \operatorname{acc}(\mathbf{P}_k, \mathbf{T}_k) \tag{6}$$

To visualize our results, we use a 40 × 40 pixel image representation. Each pixel corresponds to a specific combination of mean (μ) and standard deviation (σ) values, which characterize the distribution of scores within a sequence. The pixel's color or intensity reflects the AI's performance on sequences with those specific parameters. This representation allows us to observe how the AI's ability to sort varies with μ and σ, which directly influence the complexity of the task.

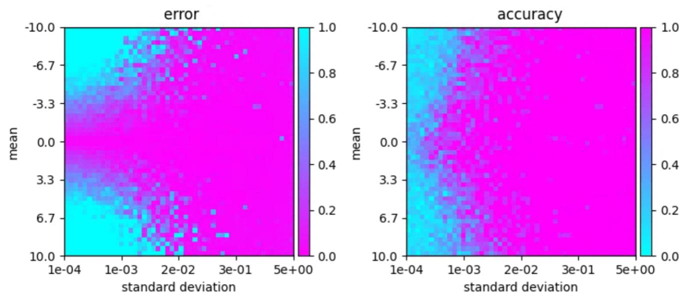

Fig. 4. The reference model is trained directly on the full dataset, using a single training step without any progressive learning. The data is sampled from the window $[\mu_{min}, \mu_{max}] \times [\sigma_{min}, \sigma_{max}] = [-10, 10] \times [0.0001, 5]$ ($r = 5 \cdot 10^{-6}$ as defined in eq. (4)), containing 500,000 sequences. Training uses an RMSE loss normalized by the output standard deviation and is performed over 1600 iterations. The left panel shows the error, and the right panel shows the accuracy of the model across the map of operating points.

For each operating point (μ,σ)—or each pixel—we draw a sequence with the corresponding score distribution and evaluate two metrics:

- The weighted sequence error, computed using eq. (1), measures the deviation of the model's predictions from the ideally sorted sequence, weighted by the natural difficulty of each sequence. This value ranges from 0 (perfect learning) to 1 (no learning).
- The sequence accuracy, computed using eq. (5), quantifies the model's ability to correctly reorder the sequence. This value ranges from 0.1 (poor abilities) to 1 (excellent abilities).

The image highlights the model's failure to learn how to correctly sort all sequences. Specifically, sequences with closely spaced scores within the same sequence (located on the left side of each image), which we considered particularly challenging due to the required precision, are almost never sorted correctly. Additionally, in the error map (on the left), we observe a region where the error is low despite the precision also being low. This region corresponds to sequences that naturally have a low error using eq. 1, and this artifact appears due to the absence of the normalization we introduce in eq. (2).

We adopt the following training strategy: the model is trained successively on 9 datasets controlled by windows of the form

$$[\mu_{min}, \mu_{max}] \times [\sigma_{min}, \sigma_{max}] = [-10, 10] \times [\lambda, 5]$$

as described in Section III. Each window contains 500,000 sequences, and training is performed for 170 iterations per window. Here, λ is the parameter controlling the minimum separability of the sequences, and it takes the values $[1,\ 3.10^{-1},\ 1.10^{-1},\ 3.10^{-2},\ 1.10^{-2},\ 3.10^{-3},\ 1.10^{-3},\ 3.10^{-4},\ 1.10^{-4}]$.

The results, illustrated in Figure 5, demonstrate that the model initially performs well on a restricted domain but gradually extends its capabilities to the entire domain as the windows (limited by the black line) are progressively introduced during training. These results stand in stark contrast to the reference model, whose good performance, shown in Figure 4, could not extend across all operating points (μ,σ), especially for challenging points.

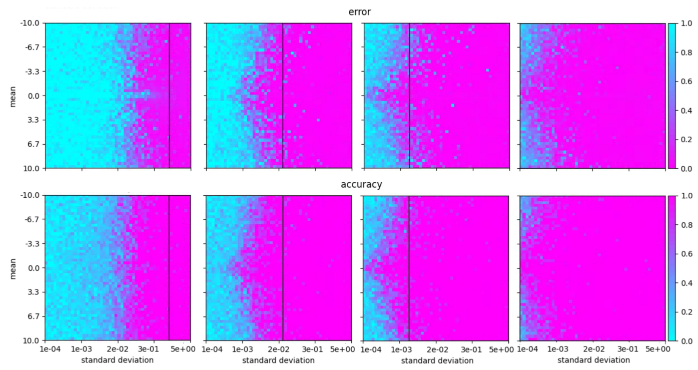

Fig. 5. Transformer performance at different stages of training: after the 2^{nd}, 4^{th}, 8^{th}, and 10^{th} training windows. Error is displayed in the top row, and accuracy in the bottom row.

We present our final results in Figure 6: The fully trained network is evaluated over the same set of windows used in Figure 2. For Transformer, we report the average performance across each window and provide boxplots to highlight the disparity in performance. The progressive training strategy facilitates gradual learning and significantly enhances final performance, particularly for the Transformer, which outperforms the CNN. While CNN results remain mitigate, they still surpass those obtained without our custom loss function and training strategy.

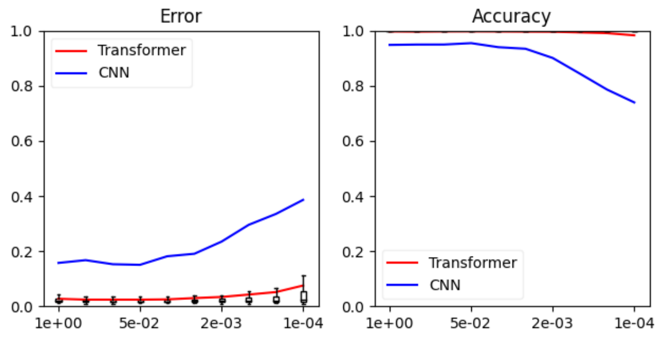

Fig. 6. Error and Accuracy Across Windows with Performance Disparity

VI. Conclusion

This finding underscores the general abilities for Transformers to deal with sequences, and highlight the potential of the approch to address challenging learning tasks across diverse architectures.

We proposed a loss function incorporating sequence weighting, along with a progressive learning strategy, which enabled us to maintain consistent AI performance across all sequence types. This approach also led to higher accuracy compared to conventional loss functions and standard training methods, as shown in Figures 6 and 2. Quantitatively, we improved the precision limit from $r = 10^{-4}$ to $r = 5.10^{-6}$. However, while our method significantly benefits the Transformer, the improvements observed for CNNs remain more modest. This discrepancy could be due to a suboptimal capacity of the CNN architecture used in our experiments. Since convolutions operate locally, capturing order dependencies may require deeper networks or larger kernel sizes to adequately model the sorting task. Further investigation would be needed to determine whether increasing the CNN's depth or receptive field could fill the performance gap with the Transformer.

References

[1] Susto, et al. "Machine Learning for Predictive Maintenance: A Multiple Classifier Approach" in *Industrial Informatics, IEEE Transactions on* vol. 11, pp. 812–820, 2015.

[2] Katoh, et al. "MAFFT Multiple Sequence Alignment Software Version 7: Improvements in Performance and Usability" in *Molecular Biology and Evolution*, vol. 30, pp. 772-–780, 2013. Available: https://doi.org/10.1093/molbev/mst010

[3] Harilal, et al. "STint: Self-supervised Temporal Interpolation for Geospatial Data," in * arXiv:2309.00059*, 2023. [Online]. Available: https://arxiv.org/abs/2309.00059.

[4] Radford, et al. "Improving Language Understanding by Generative Pre-Training", OpenAI, 2018. [Online].

[5] Goodfellow, et al. *Deep Learning*. MIT Press, 2016.

[6] Urdshals, et al. "Structure Development in List-Sorting Transformers," in * arXiv:2501.18666*, 2025. [Online]. Available: https://arxiv.org/abs/2501.18666.

[7] Kraska, et al. "The Case for Learned Index Structures," in *SIGMOD '18: Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 489–504.

[8] Zhu, et al. "Deep Learning Service for Efficient Data Distribution Aware Sorting," in *2024 IEEE International Conference on Big Data (BigData)*, 2024, pp. 1508–1515.

[9] Ferragina, et al. "Balanced Learned Sort: A New Learned Model for Fast and Balanced Item Bucketing," Available: https://arxiv.org/html/2407.00734v1, 2024.

[10] Kim, et al. "Generalized Neural Sorting Networks with Error-Free Differentiable Swap Functions," in *The Twelfth International Conference on Learning Representations*, 2024 [Online]. Available: https://openreview.net/forum?id=RLSWbk9kPw.

[11] Bagiński, "One Attention Head Is All You Need for Sorting Fixed-Length Lists," 2023. Available: https://www.apartresearch.com/project/one-attention-head-is-all-you-need-for-sorting-fixed-length-lists.

[12] Albawi, et al. "Understanding of a Convolutional Neural Network," in *2017 International Conference on Engineering and Technology (ICET)*, 2017, pp. 1–6.

[13] Vaswani, et al. "Attention Is All You Need," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 5998–6008.

[14] Wang, et al. "A Comprehensive Survey of Continual Learning: Theory, Method and Application," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 8, pp. 5362-5383, 2024.

[15] Kalchbrenner, et al. "Neural Machine Translation in Linear Time," in *arXiv preprint arXiv:1610.10099*, 2017. Available: https://arxiv.org/abs/1610.10099.