## A Appendix: Illustrations

### A.1 Baseline Architectures

Figures 5 and 6 illustrate the difference in architectures between the main baselines considered in the paper. Transformer + Cross Attention is composed of a Transformer encoder ($\mathbb{R}^{N\times D} \rightarrow \mathbb{R}^{N\times D}$) and a Cross Attention module ($\mathbb{R}^{M\times D} \times \mathbb{R}^{N\times D} \rightarrow \mathbb{R}^{M\times D}$). Perceiver IO is composed of an iterative attention encoder ($\mathbb{R}^{N\times D} \rightarrow \mathbb{R}^{L\times D}$) and a Cross Attention module ($\mathbb{R}^{M\times D} \times \mathbb{R}^{L\times D} \rightarrow \mathbb{R}^{M\times D}$). In contrast, ReTreever (Figure 1) is composed of a flexible encoder ($\mathbb{R}^{N\times D} \rightarrow \mathbb{R}^{N\times D}$) and a Tree Cross Attention module.

Empirically, we showed that Transformer + Cross Attention achieves good performance. However, its Cross Attention is inefficient due to retrieving from the full set of tokens. Notably, Perceiver IO achieves its inference-time efficiency by performing a compression via an iterative attention encoder. To achieve token-efficient inferences, the number of latents needs to be significantly less than the number of tokens originally, i.e., $L << N$. However, this is not practical in hard problems since setting low values for $L$ results in significant information loss due to the latents being a bottleneck. In contrast, ReTreever is able to perform token-efficient inference while achieving better performance than Perceiver IO for the same number of tokens. ReTreever does this by using Tree Cross Attention to retrieve the necessary tokens, only needing a logarithmic number of tokens $\log(N) << N$, making it efficient regardless of the encoder used. Crucially, this also means that ReTreever is more flexible with its encoder than Perceiver IO, allowing for customizing the kind of encoder depending on the setting.

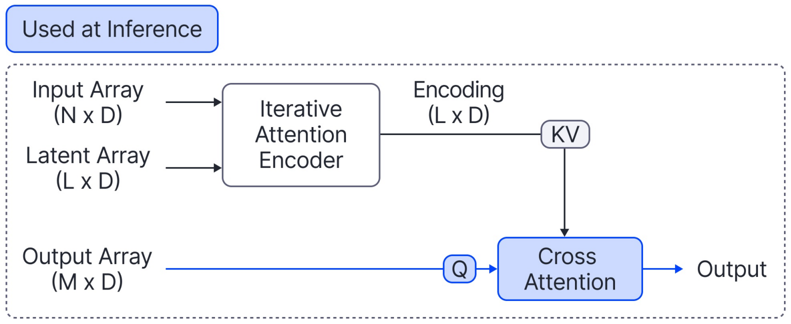

Figure 5: Architecture Diagram of Perceiver IO. The model is composed of an iterative attention encoder ($\mathbb{R}^{N\times D} \rightarrow \mathbb{R}^{L\times D}$) and a Cross Attention module ($\mathbb{R}^{M\times D} \times \mathbb{R}^{L\times D} \rightarrow \mathbb{R}^{M\times D}$).

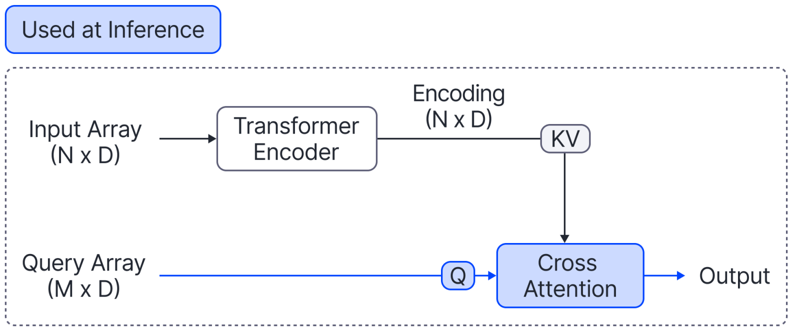

Figure 6: Architecture Diagram of Transformer + Cross Attention. The model is composed of a Transformer encoder ($\mathbb{R}^{N\times D} \rightarrow \mathbb{R}^{N\times D}$) and a Cross Attention module ($\mathbb{R}^{M\times D} \times \mathbb{R}^{N\times D} \rightarrow \mathbb{R}^{M\times D}$).

### A.2 Example Retrieval

In Figures 7 to 11, we illustrate an example of the retrieval process described in Algorithm 1.

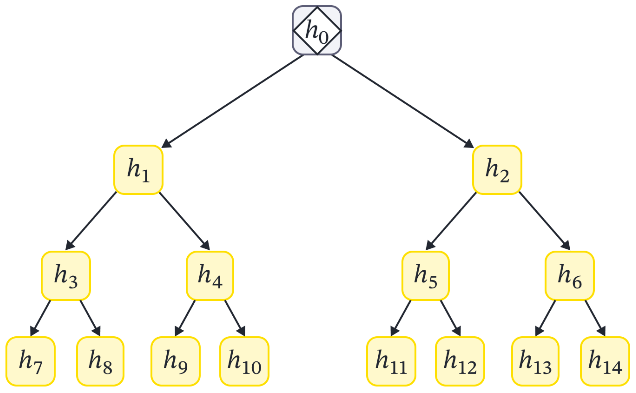

Figure 7: Example Retrieval Step 0 Illustration. The diamond icon on a node indicates the node that will be explored. The retrieval process begins at the root, i.e., $v \leftarrow 0$ where $0$ is the index of the root node. Yellow nodes denote the nodes that have yet to be explored and may be explored in the future. Grey nodes denote the nodes that are or have been explored but have not been added to the selected set $\mathbb{S}$. Red nodes denote the nodes that have not been explored and will not be explored in the future. Green nodes denote the subset of nodes selected, i.e., $\mathbb{S}$.

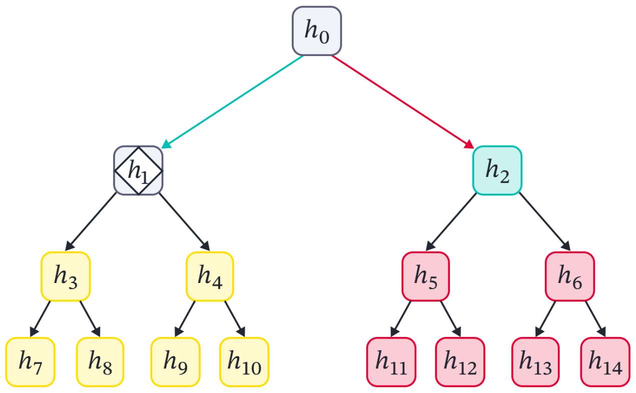

Figure 8: Example Retrieval Step 1 Illustration. Green arrows represent the path (actions) chosen by the policy $\pi$ for a query feature vector. Red arrows represent the actions rejected by the policy. The right child of $v = 0$ was rejected, so it is added to the selected set $\mathbb{S}$. As a consequence, descendant nodes of $v$'s right child will never be explored, so they are coloured red in the diagram. Tentatively, $\mathbb{S} = \{h_2\}$. Because the policy selected the left child of $v = 0$, we thus have $v \leftarrow 1$

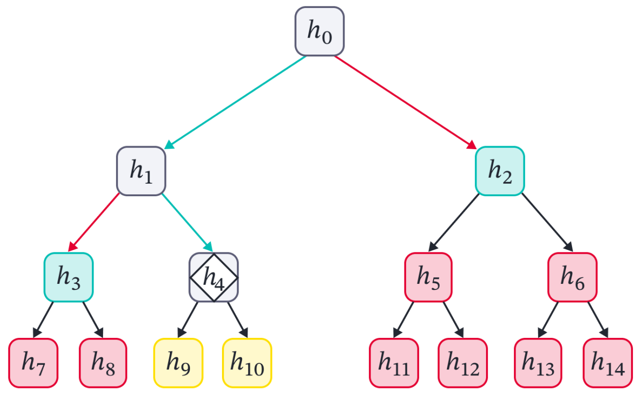

Figure 9: Example Retrieval Step 2 Illustration. The left child of $v = 1$ is rejected this time, so it is added to the selected set $\mathbb{S}$ (green). Tentatively, $\mathbb{S} = \{h_2, h_3\}$. Consequently, the descendant nodes of $v$'s left child will never be explored, so they are coloured red. Because the policy selected the right child of $v = 1$, we thus have $v \leftarrow 4$.

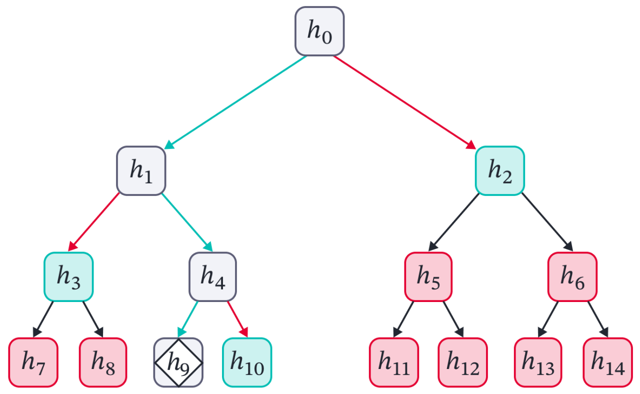

Figure 10: Example Retrieval Step 3 Illustration. The right child of $v = 4$ is rejected this time, so it is added to the selected set $\mathbb{S}$. Tentatively, $\mathbb{S} = \{h_2, h_3, h_{10}\}$. The descendant nodes of $v$'s right child are to be coloured red, but there are none. Because the policy selected the left child of $v = 4$, we thus have $v \leftarrow 9$

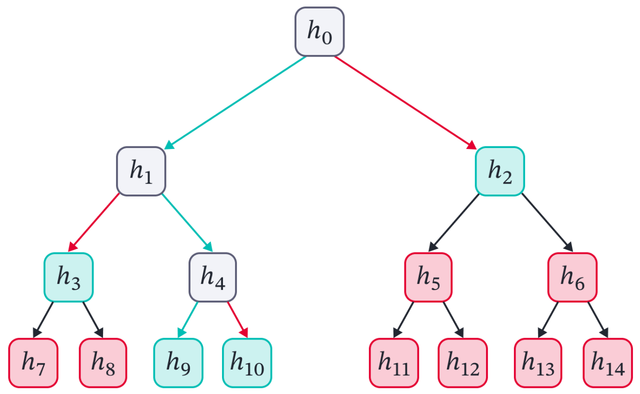

Figure 11: Example Retrieval Step 4 Illustration. $v = 9$ is a leaf, so the tree search has reached its end. At the end of the tree search, we simply add the node to the selected set, resulting in $\mathbb{S} = \{h_2, h_3, h_9, h_{10}\}$.

| Method | % Tokens | CelebA | EMNIST |
|---|---|---|---|
| Transformer + Cross Attention | 100% | $\mathbf{3.88 \pm 0.01}$ | $\mathbf{1.41 \pm 0.00}$ |
| Perceiver IO | $\mathbf{4.6\%}$ | $3.20 \pm 0.01$ | $1.25 \pm 0.00$ |
| ReTreever-Random | $\mathbf{4.6\%}$ | $3.12 \pm 0.02$ | $1.26 \pm 0.01$ |
| ReTreever-KD-$x_0$ | $\mathbf{4.6\%}$ | $3.50 \pm 0.02$ | $\mathbf{1.30 \pm 0.01}$ |
| ReTreever-KD-$x_1$ | $\mathbf{4.6\%}$ | $\mathbf{3.52 \pm 0.01}$ | $1.26 \pm 0.02$ |
| ReTreever-KD-RP | $\mathbf{4.6\%}$ | $3.51 \pm 0.01$ | $1.29 \pm 0.02$ |

Table 6: Tree Design Analyses Experiments. We compare the performance of ReTreever with various heuristics for structuring the tree (1) organized randomly (ReTreever-Random), (2) sorting via the first index $x_0$ of the image data (ReTreever-KD-$x_0$), and (3) sorting via the second index $x_1$ of the image data (ReTreever-KD-$x_1$), and (4) sorting the value after a Random Projection (ReTreever-KD-RP).

| Method | GP Regression | | | Image Completion | |
|---|---|---|---|---|---|
| | % Tokens | RBF | Matern 5/2 | % Tokens | CelebA |
| ReTreever | $\mathbf{14.9\%}$ | $1.25 \pm 0.02$ | $0.81 \pm 0.02$ | $\mathbf{4.6\%}$ | $3.52 \pm 0.01$ |
| ReTreever-Full | 100% | $\mathbf{1.33 \pm 0.02}$ | $\mathbf{0.90 \pm 0.01}$ | 100% | $\mathbf{3.77 \pm 0.01}$ |

Table 7: Comparison of ReTreever with ReTreever-Full which naively selects all the leaves of the tree instead of leveraging the learned policy to select a subset of the nodes.

## B Appendix: Additional Experiments and Analyses

### B.1 Additional Analyses

#### B.1.1 Importance of Tree Design

Unlike sequential data where sorting according to its index is a natural way to organize the data, it is less clear how to organize other data modalities (e.g., pixels) as leaves in a balanced tree. As such, in this experiment (Table 6), we compare the performance of ReTreever with various heuristics for structuring the tree (1) organized randomly (ReTreever-Random), (2) sorting via the first index $x_0$ of the image data (ReTreever-KD-$x_0$), and (3) sorting via the second index $x_1$ of the image data (ReTreever-KD-$x_1$), and (4) sorting the value after a Random Projection (ReTreever-KD-RP)[4]. ReTreever-KD-RP generates a random matrix $w \in \mathbb{R}^{dim(x) \times 1}$ that is fixed during training and orders the context tokens in the tree according to $w^T x$ instead.

In Table 6, we see that ReTreever outperforms Perceiver IO significantly when splitting according to either of the axes. In contrast, organizing the tree randomly causes ReTreever to perform worse than Perceiver IO. There is, however, a minor difference between KD-$x_0$ and KD-$x_1$. We hypothesize that this is in part due to image data being multi-dimensional and not all dimensions are equal.

#### B.1.2 ReTreever-Full

ReTreever leverages RL to learn good node representations for informative retrieval. However, ReTreever is not limited to only using the learned policy for retrieval at inference time. For example, instead of retrieving a subset of the nodes according to a policy, we can (as a proof of concept) choose to retrieve all the leaves of the tree "ReTreever-Full" (see Table 7).

**Results.** On CelebA and GP Regression, we see that ReTreever-Full achieves performance close to state-of-the-art. For example, on CelebA, ReTreever-Full achieves $3.77 \pm 0.01$, and Transformer + Cross Attention achieves $3.88 \pm 0.01$. On GP, ReTreever-Full achieves $1.33 \pm 0.02$, and Transformer + Cross Attention achieves $1.35 \pm 0.02$. In contrast, ReTreever achieves significantly lower results. The performance gap between ReTreever and ReTreever-Full suggests that the performance of ReTreever can vary depending on the number of nodes (tokens) selected from the tree. As such, an interesting future direction is the design of alternative methods for selecting subsets of nodes.

[4]Random Projection is popular in dimensionality reduction literature as it preserves partial information regarding the distances between the points.

| Method | $N = 256$ | | $N = 512$ | | $N = 1024$ | |
|---|---|---|---|---|---|---|
| | % Tokens | Accuracy | % Tokens | Accuracy | % Tokens | Accuracy |
| Cross Attention | 100.0% | $\mathbf{100.0 \pm 0.0}$ | 100.0% | $\mathbf{100.0 \pm 0.0}$ | 100.0% | $\mathbf{99.9 \pm 0.2}$ |
| Random | — | $8.3 \pm 0.0$ | — | $8.3 \pm 0.0$ | — | $8.3 \pm 0.0$ |
| Perceiver IO | $\mathbf{6.3}\%$ | $15.2 \pm 0.0$ | $\mathbf{3.5}\%$ | $13.4 \pm 0.2$ | $\mathbf{2.0}\%$ | $11.6 \pm 0.4$ |
| Perceiver IO | 100.0% | $63.6 \pm 45.4$ | 100.0% | $14.1 \pm 2.3$ | 100.0% | OOM |
| TCA | $\mathbf{6.3}\%$ | $\mathbf{100.0 \pm 0.0}$ | $\mathbf{3.5}\%$ | $\mathbf{100.0 \pm 0.0}$ | $\mathbf{2.0}\%$ | $\mathbf{99.6 \pm 0.6}$ |

Table 8: Copy Task Results with accuracy (higher is better) and $\%$ tokens (lower is better) metrics.

| Perceiver IO Num. Latents ($L$) | Copy Task | | |
|---|---|---|---|
| | $N = 128$ | $N = 256$ | $N = 512$ |
| $L = 16$ | 39.83 ± 40.21 | 15.16 ± 0.03 | 13.41 ± 0.15 |
| $L = 32$ | 58.92 ± 47.44 | 34.75 ± 39.14 | 19.41 ± 12.07 |
| $L = 64$ | 79.45 ± 41.11 | 27.10 ± 21.23 | 14.00 ± 0.72 |
| $L = 128$ | 100.00 ± 0.00 | 63.50 ± 39.37 | 13.23 ± 0.42 |

Table 9: Analysis of Perceiver IO's performance relative to the number of latents ($L$) and the sequence length ($N$) on the Copy Task

#### B.1.3 Perceiver IO's Copy Task Performance

In Table 8, we include results evaluating Perceiver IO with increased number of latent tokens on varying lengths of the copy task. Specifically, we tested using $100\%$ tokens to evaluate its performance. Perceiver IO's performance degraded sharply as the difficulty of the task increased. Furthermore, due to Perceiver IO's encoder which aims to map the context tokens to a set of latent tokens, we ran out of memory (on a Nvidia P100 GPU (16 GB)) trying to train the model on a sequence length of 1024.

To analyze the reason behind the degradation in performance of Perceiver IO, we evaluated Perceiver IO (Table 9) with a varying number of latent tokens $L \in \{16, 32, 64, 128\}$ and varying sequence lengths $N \in \{128, 256, 512\}$. Generally, we found that performance improved as the number of latents increased. However, the results were relatively unstable. For some runs, Perceiver IO was able to solve the task completely. Other times, Perceiver IO got stuck in lower accuracy local optimas. For longer sequences $N = 512$, increasing the number of latents did not make a difference as the model simply got stuck in poor local optimas.

We hypothesize that the poor performance is due to the incompatibility between the nature of the Copy Task and Perceiver IO's model. The copy task (1) has high intrinsic dimensionality that scales with the number of tokens and (2) does not require computing higher-order information between the tokens. In contrast, Perceiver IO (1) is designed for tasks with lower intrinsic dimensionality by compressing information via its latent tokens and iterative attention encoder, and (2) computes higher-order information via a transformer in the latent space. Naturally, these factors make learning the task difficult for Perceiver IO.

#### B.1.4 Runtime Analyses

**Runtime of Tree Construction and Aggregation.** For 256 tokens, the wall-clock time for constructing a tree with our implementation was $0.076 \pm 0.022$ milliseconds. The wall-clock time for the aggregation is $12.864 \pm 2.792$ milliseconds.

**Runtime of Tree Cross Attention-based models vs. vanilla Cross Attention-based models.** In table 10, we measured the wall-clock time of the modules used during inference time, i.e., ReTreever's Tree Cross Attention module ($\mathbb{R}^{M\times D} \times \mathcal{T} \to \mathbb{R}^{M\times D}$), Transformer+Cross Attention's Cross Attention module ($\mathbb{R}^{M\times D} \times \mathbb{R}^{N\times D} \to \mathbb{R}^{M\times D}$), and Perceiver IO's Cross Attention module ($\mathbb{R}^{M\times D} \times \mathbb{R}^{L\times D} \to \mathbb{R}^{M\times D}$).

We evaluated the modules on the Copy Task for both a GPU and CPU. Although Tree Cross Attention is slower, all methods only require milliseconds to perform inference. Both Tree Cross Attention and Cross Attention learn to solve the task perfectly. However, Tree Cross Attention uses

| Model | % Tokens | Accuracy | CPU time | GPU Time |
|---|---|---|---|---|
| Cross Attention | 100.0% | **100.0 ± 0.0** | 12.05 | 1.61 |
| Tree Cross Attention | **3.5%** | **100.0 ± 0.0** | 19.31 | 9.09 |
| Perceiver IO's CA | **3.5%** | 13.4 ± 0.2 | **10.98** | **1.51** |

Table 10: Comparison of runtime (in milliseconds) for the inference modules of ReTreever, Transformer + Cross Attention, and Perceiver IO, i.e., Tree Cross Attention, Cross Attention, and Perceiver IO's CA respectively. Runtime is reported as the average of 10 runs.

| Model | Tree Height ($H$) | % Tokens | CPU Time (in ms) | GPU Time (in ms) |
|---|---|---|---|---|
| Cross Attention | N/A | 100.0 % | 12.05 | 1.61 |
| TCA ($b_f = 256$) | 1 | 100.0 % | 16.21 | 2.05 |
| TCA ($b_f = 32$) | 2 | 15.2 % | 14.05 | 3.99 |
| TCA ($b_f = 16$) | 2 | 12.1 % | 13.25 | 4.13 |
| TCA ($b_f = 8$) | 3 | 7.0 % | 15.53 | 5.73 |
| TCA ($b_f = 4$) | 3 | 3.9 % | 16.30 | 6.45 |
| TCA ($b_f = 2$) | 8 | 3.5 % | 21.92 | 9.83 |

Table 11: Memory-Runtime trade-off with respect to the tree height ($H$).

only $3.5\%$ of the tokens. As such, Tree Cross Attention tradesoff between the number of tokens and computational time. Perceiver IO, however, fails to solve the task for the same number of tokens.

**The effect of tree design on Memory-Runtime trade-off.** Tree Cross Attention sequentially searches down a tree for the relevant tokens. As such, the factor which primarily affects the overall runtime is the height of tree ($H$) which is dependent on the branching factor $b_f \geq 2$. The relationship between the height of the tree, the branching factor, and the number of tokens is as follows: $H = \lceil \log_{b_f}(N) \rceil$. In our results, we showed the performance by using a binary tree ($b_f = 2$) since this corresponds to a tree design that uses few tokens. By selecting a higher branching factor, the runtime can be decreased at the expense of requiring more tokens (see Table 11). This feature is not available in standard cross attention. As such, TCA enables more control over the memory and computation time trade-off.

## B.2 Additional Experiments

### B.2.1 Copy Task

| Method | $N = 128$ | |
|---|---|---|
| | % Tokens | Accuracy |
| Cross Attention | $100.0\%$ | $\mathbf{100.0 \pm 0.0}$ |
| Random | — | $8.3 \pm 0.0$ |
| Perceiver IO | $\mathbf{10.9}\%$ | $17.8 \pm 0.0$ |
| Perceiver IO | $\mathbf{100.0}\%$ | $100.0 \pm 0.0$ |
| TCA (Acc) | $\mathbf{10.9}\%$ | $\mathbf{100.0 \pm 0.0}$ |
| TCA (CE) | $\mathbf{10.9}\%$ | $\mathbf{100.0 \pm 0.0}$ |

Table 12: Copy Task Results with accuracy (higher is better) and $\%$ tokens (lower is better) metrics.

**Results.** In Table 12, we include additional results for the Copy Task where $N = 128$. Similar to previous results, we see that Cross Attention and TCA are able to solve this task perfectly. In contrast, Perceiver IO is not able to solve this task for the same number of tokens. We also see both TCA using accuracy and TCA using negative cross entropy as the reward were able to solve the task perfectly.

| Method | % Tokens | RBF | Matern 5/2 |
|---|---|---|---|
| CNP (Garnelo et al., 2018a) | — | 0.26 ± 0.02 | 0.04 ± 0.02 |
| CANP (Kim et al., 2019) | — | 0.79 ± 0.00 | 0.62 ± 0.00 |
| NP (Garnelo et al., 2018b) | — | 0.27 ± 0.01 | 0.07 ± 0.01 |
| ANP (Kim et al., 2019) | — | 0.81 ± 0.00 | 0.63 ± 0.00 |
| BNP (Lee et al., 2020) | — | 0.38 ± 0.02 | 0.18 ± 0.02 |
| BANP (Lee et al., 2020) | — | 0.82 ± 0.01 | 0.66 ± 0.00 |
| TNP-D (Nguyen & Grover, 2022) | — | $\mathbf{1.39 \pm 0.00}$ | $\mathbf{0.95 \pm 0.01}$ |
| LBANP (Feng et al., 2023a) | — | 1.27 ± 0.02 | 0.85 ± 0.02 |
| CMANP (Feng et al., 2023b) | — | 1.24 ± 0.02 | 0.80 ± 0.01 |
| Perceiver IO ($L = 4$) | 8.5% | 1.02 ± 0.03 | 0.56 ± 0.03 |
| Perceiver IO ($L = 8$) | 17.0% | 1.13 ± 0.03 | 0.65 ± 0.03 |
| Perceiver IO ($L = 16$) | 34.0% | 1.22 ± 0.05 | 0.75 ± 0.06 |
| Perceiver IO ($L = 32$) | 68.0% | 1.25 ± 0.04 | 0.78 ± 0.05 |
| Perceiver IO ($L = 64$) | 136.0% | 1.30 ± 0.03 | 0.85 ± 0.02 |
| Perceiver IO ($L = 128$) | 272.0% | 1.29 ± 0.04 | 0.84 ± 0.04 |
| Transformer + Cross Attention | 100% | $\mathbf{1.35 \pm 0.02}$ | $\mathbf{0.91 \pm 0.02}$ |
| Perceiver IO | 14.9% | 1.06 ± 0.05 | 0.58 ± 0.05 |
| ReTreever | 14.9% | 1.25 ± 0.02 | 0.81 ± 0.02 |

Table 13: 1-D Meta-Regression Experiments with log-likelihood metric (higher is better).

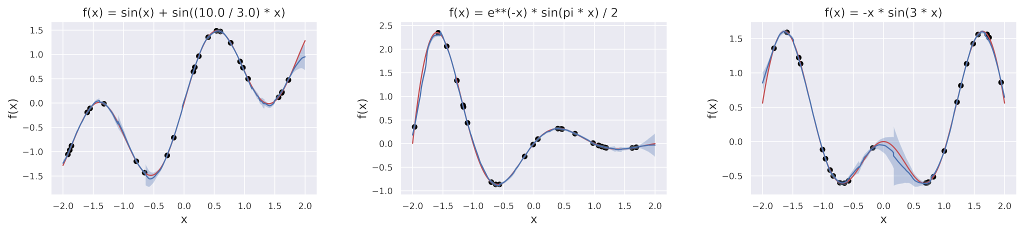

Figure 12: Visualizations of GP Regression

| Method | % Tokens | CelebA |
|---|---|---|
| CNP | — | 2.15 ± 0.01 |
| CANP | — | 2.66 ± 0.01 |
| NP | — | 2.48 ± 0.02 |
| ANP | — | 2.90 ± 0.00 |
| BNP | — | 2.76 ± 0.01 |
| BANP | — | 3.09 ± 0.00 |
| TNP-D | — | 3.89 ± 0.01 |
| LBANP (8) | — | 3.50 ± 0.05 |
| LBANP (128) | — | 3.97 ± 0.02 |
| CMANP | — | 3.93 ± 0.05 |
| Perceiver IO ($L = 8$) | 4.1% | 3.18 ± 0.03 |
| Perceiver IO ($L = 16$) | 8.1% | 3.35 ± 0.02 |
| Perceiver IO ($L = 32$) | 16.2% | 3.50 ± 0.02 |
| Perceiver IO ($L = 64$) | 32.5% | 3.61 ± 0.03 |
| Perceiver IO ($L = 128$) | 65.0% | 3.74 ± 0.02 |
| Transformer + Cross Attention | 100% | $\mathbf{3.88 \pm 0.01}$ |
| Perceiver IO | 4.6% | $3.20 \pm 0.01$ |
| ReTreever | 4.6% | $3.52 \pm 0.01$ |

Table 14: CelebA Image Completion Experiments. The methods are evaluated according to the log-likelihood (higher is better).

### B.2.2 GP Regression

**Results.** In Table 13, we show results for all baselines on GP Regression. We see that Transformer + Cross Attention performs comparable to the state-of-the-art, outperforming all Neural Process baselines except for TNP-D by a significant margin. We evaluated Perceiver IO with varying number of latents $L$. We see that Perceiver IO ($L = 32$) uses $\sim 4.6\times$ the number of tokens to achieve performance comparable to ReTreever. ReTreever by itself already outperforms all NP baselines except for LBANP and TNP-D. Visualizations for different kinds of test functions are included in Figure 12.

### B.2.3 Image Completion (CelebA)

**Results.** In Table 14, we show results for all baselines on CelebA. We see that Transformer + Cross Attention achieves results competitive with the state-of-the-art. We evaluated Perceiver IO with varying numbers of latents $L$. We see that Perceiver IO ($L = 32$) uses $\sim 3.5\times$ the number of tokens to achieve performance comparable to the ReTreever. Visualizations are available in Figure 13.

### B.2.4 Image Completion (EMNIST)

**Results.** In Table 15, we see that Transformer + Cross Attention achieves results competitive with state-of-the-art, outperforming all baselines except TNP-D.

### B.2.5 Time Series: Human Activity

**Results.** In Table 16, we show results on the Human Activity task comparing with several time series baselines. We see that Transformer + Cross Attention outperforms all baselines except for mTAND-Enc. The confidence intervals are, however, very close to overlapping. We would like to note, however, that mTAND-Enc was carefully designed for time series, combining attention, bidirectional RNNs, and reference points. In contrast, Transformer + Cross Attention is a general-purpose model made by just leveraging simple attention modules. We hypothesize the performance could be further improved by leveraging some features of mTAND-Enc, but this is outside the scope of our work.

Notably, ReTreever also outperforms all baselines except for mTAND-Enc. Comparing, Transformer + Cross Attention and ReTreever, we see that ReTreever achieves comparable performance

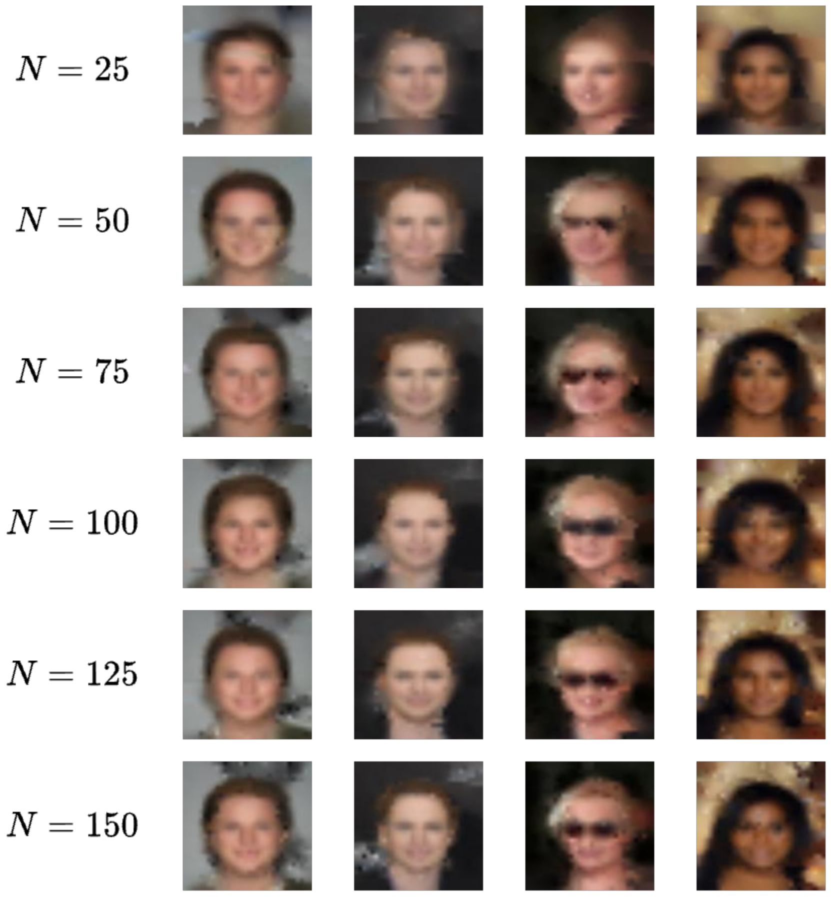

Figure 13: Visualizations of CelebA32

| Method | % Tokens | EMNIST |
|---|---|---|
| CNP | — | 0.73 ± 0.00 |
| CANP | — | 0.94 ± 0.01 |
| NP | — | 0.79 ± 0.01 |
| ANP | — | 0.98 ± 0.00 |
| BNP | — | 0.88 ± 0.01 |
| BANP | — | 1.01 ± 0.00 |
| TNP-D | — | 1.46 ± 0.01 |
| LBANP (8) | — | 1.34 ± 0.01 |
| LBANP (128) | — | 1.39 ± 0.01 |
| CMANP | — | 1.36 ± 0.01 |
| Transformer + Cross Attention | 100% | 1.41 ± 0.00 |
| Perceiver IO | 4.6% | 1.25 ± 0.00 |
| ReTreever | 4.6% | 1.30 ± 0.01 |

Table 15: EMNIST Image Completion Experiments. The methods are evaluated according to the log-likelihood (higher is better).

| Model | % Tokens | Accuracy |
|---|---|---|
| RNN-Impute (Che et al., 2018) | — | 85.9 ± 0.4 |
| RNN-Decay (Che et al., 2018) | — | 86.0 ± 0.5 |
| RNN GRU-D (Che et al., 2018) | — | 86.2 ± 0.5 |
| IP-Nets (Shukla & Marlin, 2019) | — | 86.9 ± 0.7 |
| SeFT (Horn et al., 2020) | — | 81.5 ± 0.2 |
| ODE-RNN (Rubanova et al., 2019) | — | 88.5 ± 0.8 |
| L-ODE-RNN (Chen et al., 2018) | — | 83.8 ± 0.4 |
| L-ODE-ODE (Rubanova et al., 2019) | — | 87.0 ± 2.8 |
| mTAND-Enc (Shukla & Marlin, 2021) | — | 90.7 ± 0.2 |
| Transformer + Cross Attention | 100% | 89.1 ± 1.3 |
| Perceiver IO | 14% | 87.6 ± 0.4 |
| ReTreever | 14% | 88.9 ± 0.4 |

Table 16: Human Activity Experiments with Accuracy metric.

while using far fewer tokens. Furthermore, we see that ReTreever outperforms Perceiver IO significantly while using the same number of tokens.

## C Appendix: Implementation and Hyperparameters Details

### C.1 Implementation

For the experiments on uncertainty estimation, we use the official repositories for TNPs (https://github.com/tung-nd/TNP-pytorch) and LBANPs (https://github.com/BorealisAI/latent-bottlenecked-anp). The GP regression and Image Classification (CelebA and EMNIST) datasets are available in the same links. For the Human Activity dataset, we use the official repository for mTAN (Multi-Time Attention Networks) https://github.com/reml-lab/mTAN. The Human Activity dataset is available in the same link. Our Perceiver IO baseline is based on the popular Perceiver (IO) repository (https://github.com/lucidrains/perceiver-pytorch/blob/main/perceiver_pytorch/perceiver_pytorch.py). We report the baseline results listed in Nguyen & Grover (2022) and Shukla & Marlin (2021). When comparing methods such as Perceiver, ReTreever, Transformer + Cross Attention, LBANP, and TNP-D, we use the same number of blocks (i.e., CMABs, iterative attention, and transformer) in the encoder.

### C.2 Hyperparameters

Following previous works on Neural Processes (LBANPs and TNPs), ReTreever uses 6 layers in the encoder for all experiments. Our aggregator function is a Self Attention (Transformer) module whose output is averaged. Following standard RL practices, at test time, TCA and ReTreever's policy selects the most confident actions: $u = \text{argmax}_{a \in \mathbb{C}_v} \pi_\theta(a|s)$ We tuned $\lambda_{RL}$ (RL loss term weight) and $\lambda_{CA}$ (CA loss term weight) between $\{0.0, 0.1, 1.0, 10.0\}$ on the Copy Task (see analysis in main paper) with $N = 256$. We also tuned $\alpha$ (entropy bonus weight) between $\{0.0, 0.01, 0.1, 1.0\}$. We found that simply setting $\lambda_{RL} = \lambda_{CA} = 1.0$ and $\alpha = 0.01$ performed well in practice on the Copy Task where $N = 256$. As such, for the purpose of consistency, we set $\lambda_{RL} = \lambda_{CA} = 1.0$ and $\alpha = 0.01$ was set for all tasks. We used an ADAM optimizer with a standard learning rate of $5e - 4$. All experiments were run with 3 seeds. Cross Attention and Tree Cross Attention tended to get stuck in a local optimum on the Copy Task, so to prevent this, dropout was set to $0.1$ for the Copy Task for all methods. For image data, we selected a single axis to split the data according to the k-d tree. In the case of EMNIST, the data was split according to the first axis. In the case of CelebA, the data was split according to the second axis.

### C.3 Compute

Our experiments were run using a mix of Nvidia GTX 1080 Ti (12 GB) or Nvidia Tesla P100 (16 GB) GPUs. GP regression experiments took $\sim 2.5$ hours. EMNIST experiments took $\sim 1.75$ hours. CelebA experiments took $\sim 15$ hours. Human Activity experiments took $\sim 0.75$ hours.