Memory Efficient Neural Processes via Constant Memory Attention Block

Leo Feng [1 2] **Frederick Tung** [2] **Hossein Hajimirsadeghi** [2] **Yoshua Bengio** [1] **Mohamed Osama Ahmed** [2]

Abstract

Neural Processes (NPs) are popular meta-learning methods for efficiently modelling predictive uncertainty. Recent state-of-the-art methods, however, leverage expensive attention mechanisms, limiting their applications, particularly in low-resource settings. In this work, we propose Constant Memory Attentive Neural Processes (CMANPs), an NP variant that only requires **constant** memory. To do so, we first propose an efficient update operation for Cross Attention. Leveraging the update operation, we propose Constant Memory Attention Block (CMAB), a novel attention block that (i) is permutation invariant, (ii) computes its output in constant memory, and (iii) performs constant computation updates. Finally, building on CMAB, we detail Constant Memory Attentive Neural Processes. Empirically, we show CMANPs achieve state-of-the-art results on popular NP benchmarks while being significantly more memory efficient than prior methods.

1. Introduction

Neural Processes (NPs) are a popular family of meta-learning models for uncertainty estimation. They are particularly useful in low-resource settings due to not requiring retraining from scratch given new data. Over recent years, NPs have been applied to a wide variety of settings such as graph link prediction (Liang & Gao, 2022), continual learning (Requeima et al., 2019), and recommender systems (Lin et al., 2021). With the growing popularity of low-memory/compute domains (e.g., IoT devices and mobile phones), deployed models in these low-resource settings must be memory efficient. Furthermore, memory access is energy intensive (Li et al., 2022a). As such, the design of memory-efficient models is crucial for settings with a limited energy supply (e.g., battery-powered devices such as mobile robots).

However, the state-of-the-art NP methods (Nguyen & Grover, 2022; Feng et al., 2023) are attention-based methods that require substantial amounts of memory, limiting their applicability. For example, Transformer Neural Processes (TNPs) (Nguyen & Grover, 2022) leverage Transformers (Vaswani et al., 2017). As such, TNPs are computationally expensive, scaling quadratically with the size of the context and query dataset. Latent Bottlenecked Attentive Neural Processes (LBANPs) (Feng et al., 2023) leverage Perceiver's iterative attention (Jaegle et al., 2021b) and require $\mathcal{O}(k|\mathcal{D}_C|)$ memory where $|\mathcal{D}_C|$ is the size of the context dataset and k is a hyperparameter that scales with the difficulty of the task and the context dataset size.

Tackling this issue, we propose Constant Memory Attentive Neural Processes (CMANPs), a novel attention-based NP that is competitive with prior state-of-the-art while only requiring a constant amount of memory. To do so, we first (1) improve the computational efficiency of Cross Attention and (2) propose a novel efficient attention block.

More specifically, we propose (1) an exact update operation for Cross Attention that allows it to be efficiently updated with new context data (Section 3.1.1). Using the efficient updates operation, we show that Cross Attention can also be computed memory-efficiently (Section 3.1.2). Leveraging the aforementioned efficiency properties of Cross Attention, we propose (2) Constant Memory Attention Block (CMAB) (Section 3.2), a novel attention block, that (i) is permutation invariant, (ii) computes its output in constant memory, and (iii) performs updates in constant computation.

Finally, building on CMABs, we propose Constant Memory Attentive Neural Processes (CMANPs) (Section 3.3). By using CMABs, CMANPs (i) only require constant memory, making it naturally scalable in the number of data points and (ii) allow for updates to the context to be performed efficiently unlike prior state-of-the-art NPs. Leveraging the efficient updates property, we further introduce an Autoregressive Not-Diagonal extension (Section 3.3.1) which only requires constant memory unlike the quadratic memory required by all prior Not-Diagonal extensions of NPs. In the experiments, CMANPs achieve state-of-the-art results on popular NP benchmarks while being significantly more

[*]Equal contribution [1]Mila – Université de Montréal, Canada [2]Borealis AI, Canada. Correspondence to: Leo Feng <leo.feng@mila.quebec>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

memory efficient than prior state-of-the-art methods.

2. Background

2.1. Meta-learning for Predictive Uncertainty Estimation

In meta-learning for predictive uncertainty estimation, models are trained on a distribution of tasks $\Omega(\mathcal{T})$ to model a probabilistic predictive distribution. A task $\mathcal{T}$ is a tuple ($\mathcal{X}$, $\mathcal{Y}$, $\mathcal{L}$, q) where $\mathcal{X}, \mathcal{Y}$ are the input and output space respectively, $\mathcal{L}$ is the task-specific loss function, and $q(x, y)$ is the task-specific distribution over data points. During each meta-training iteration, B tasks $\mathbf{T} = \{\mathcal{T}_i\}_{i=1}^{B}$ are sampled from the task distribution $\Omega(\mathcal{T})$. For each task $\mathcal{T}_i \in \mathbf{T}$, a context dataset $\mathcal{D}_C^i = \{(x, y)^{i,j}\}_{j=1}^{N}$ and a target dataset $\mathcal{D}_T^i = \{(x, y)^{i,j}\}_{j=1}^{M}$ are sampled from the task-specific data point distribution $q_{\mathcal{T}_i}$. N is a fixed number of context data points and M is a fixed number of target data points. The model is adapted using the context dataset. Afterwards, the target dataset is used to evaluate the effectiveness of the adaptation and adjust the adaptation rule accordingly.

2.2. Neural Processes

Neural Processes (NPs) (Garnelo et al., 2018b) are meta-learned models that define a family of conditional distributions. Specifically, NPs condition on an arbitrary amount of context data points (labelled data points) and make predictions for a batch of target data points, while preserving invariance in the ordering of the context dataset. NPs consist of three phases: conditioning, querying, and updating. However, recent state-of-the-art NP models (Nguyen & Grover, 2022; Feng et al., 2023) leverage attention, causing these phases to require large amounts of memory and limiting their applicability.

Conditioning: In the conditioning phase, the model encodes the context dataset $\mathcal{D}_C$. Neural Processes model functional uncertainty by encoding the context dataset as a Gaussian latent variable: $z_C \sim q(z|\mathcal{D}_C)$ where $q(z|\mathcal{D}_C) = \mathcal{N}(z; \mu_C, \sigma_C^2)$ and $\mu_C, \sigma_C = f_{encoder}(\mathcal{D}_C)$. Conditional variants (Garnelo et al., 2018a) instead compute a deterministic encoding, i.e., $z_C = f_{encoder}(\mathcal{D}_C)$.

Querying: In the querying phase, given target data points x_T to make predictions for, the NP models the predictive distribution $p(y_T|x_T, z_C)$.

Updating: In the updating phase, the model receives new context data points $\mathcal{D}_U$ and uses it to compute new encodings, i.e., re-computing z_C given $\mathcal{D}_C \leftarrow \mathcal{D}_C \cup \mathcal{D}_U$.

During training, deterministic NP variants maximize the log-likelihood directly. In contrast, stochastic variants maximize an evidence lower bound (ELBO) of the log-likelihood:

$$\log p(y_T|x_T, \mathcal{D}_C) \geq \mathbb{E}_{q(z|\mathcal{D}_C \cup \mathcal{D}_T)}\left[\log p(y_T|x_T, z)\right] - \mathrm{KL}(q(z|\mathcal{D}_C \cup \mathcal{D}_T)||p(z|\mathcal{D}_C))$$

3. Methodology

In this section, we propose Constant Memory Attentive Neural Processes (CMANPs), a novel attention-based NP that only requires constant memory for the conditioning, querying, and updating phases. To do so, we begin by proposing an efficient update operation for Cross Attention (Section 3.1). Leveraging the update operation, we propose Constant Memory Attention Block (CMAB) (Section 3.2), a memory-efficient attention block. Finally, building on CMAB, we propose Constant Memory Attentive Neural Processes (Section 3.3).

3.1. Cross Attention

Cross Attention (CA) is widely used in state-of-the-art Neural Processes during the conditioning, querying, and updating phases and makes up a significant portion of their computational complexity. Cross Attention retrieves information from a set of $\mathcal{D}_C$ context tokens for a given set of query tokens L via a weighted average as follows:

$$\mathrm{CrossAttention}(L, \mathcal{D}_C) = \mathrm{softmax}(QK^T)V$$

where $Q = LW_q$ is the query matrix, $K = \mathcal{D}_C W_k$ is the key matrix, and $V = \mathcal{D}_C W_v$ is the value matrix. $W_q, W_k, W_v \in \mathbb{R}^{d \times d}$ are weight matrices (learned parameters) that project the input tokens. $\mathrm{softmax}(QK^T)$ computes the weight for each context token in the weighted average.

When given a new set of context tokens $\mathcal{D}_U$, computing the updated output $\mathrm{CrossAttention}(L, \mathcal{D}_C \cup \mathcal{D}_U)$ from $\mathrm{CrossAttention}(L, \mathcal{D}_C)$ requires $\mathcal{O}((|\mathcal{D}_C| + |\mathcal{D}_U|)|L|)$ memory and computation. In practice, $|\mathcal{D}_C|$ is significantly larger than $|\mathcal{D}_U|$ and $|L|$, making the complexity's reliance on $|\mathcal{D}_C|$ a limitation.

In this section, we propose an efficient update operation that computes $\mathrm{CrossAttention}(L, \mathcal{D}_C \cup \mathcal{D}_U)$ from $\mathrm{CrossAttention}(L, \mathcal{D}_C)$ in only $\mathcal{O}(|\mathcal{D}_U||L|)$ computation:

$$\mathrm{CrossAttention}(L, \mathcal{D}_C \cup \mathcal{D}_U) = \mathrm{UPDATE}(\mathcal{D}_U, \mathrm{CrossAttention}(L, \mathcal{D}_C))$$

Leveraging this efficient update operation, we then show that $\mathrm{CrossAttention}(L, \mathcal{D}_C)$ can be efficiently computed in only $\mathcal{O}(|L|)$ memory. Notably, these complexities are independent of $\mathcal{D}_C$ making them computationally efficient and naturally scalable with respect to the number of context tokens $|\mathcal{D}_C|$. A formal proof and derivation of the efficient update operation is included in Appendix A.1.

3.1.1. Efficient Update Property

When computing the updated value of $\text{CA}' = \text{CrossAttention}(L, \mathcal{D}_C \cup \mathcal{D}_U)$ with a new set of $\mathcal{D}_U$ context tokens, $|\mathcal{D}_U|$ rows are added to the key, value matrices. Since the attention weights are computed via products between the keys (with $|\mathcal{D}_U|$ added rows) and queries (unchanged), thus the new output can be computed from $\text{CA} = \text{CrossAttention}(L, \mathcal{D}_C)$ via a rolling average in $\mathcal{O}(|\mathcal{D}_U||L|)$. An implementation of the proposed UPDATE operation for $j \in \{1, 2, \dots, |L|\}$ is as follows:

$$\text{CA}'_j = \frac{c_j}{c'_j} \times \text{CA}_j + \sum_{i=|\mathcal{D}_C|+1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \frac{e^{s_i}}{c'_j} v_i$$

where $s_i = Q_{j,:}(K_{i,:})^T$, $v_i = V_{i,:}$, and $c'_j = c_j + \sum_{i=|\mathcal{D}_C|+1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \exp(s_i)$. c' are normalizing constants for CA' that are computed via a rolling sum from cached normalizing constants c. When computing the updated value CA', c' replaces c as the cached normalizing constants. Computing CA'_j and c'_j via these rolling average and summation thus only requires $\mathcal{O}(|\mathcal{D}_U|)$ operations when given CA_j and c_j. Since there are $|L|$ values for j, the total amount of computation is thus $\mathcal{O}(|\mathcal{D}_U||L|)$. In practice, however, this is not a stable implementation. The computation of $c'_j = c_j + \sum_{i=|\mathcal{D}_C|+1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \exp(s_i)$ can quickly run into numerical issues such as underflow and overflow due to being a sum of exponentials.

As such, instead of computing c' in the update, we propose to compute $\log(c')$, resulting in the following update:

$$\text{CA}'_j = \exp(\log(c_j) - \log(c'_j)) \times \text{CA}_j + \sum_{i=|\mathcal{D}_C|+1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \exp(s_i - \log(c'_j)) v_i$$

where $\log(c'_j) = \log(c_j) + \text{softplus}(t_j)$ and $t_j = \log(\sum_{i=|\mathcal{D}_C|+1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \exp(s_i - \log(c)))$. t_j can be computed accurately using the log-sum-exp trick, avoiding the underflow and overflow issues.

3.1.2. Reducing Memory Usage

A naive computation of $\text{CrossAttention}(L, \mathcal{D}_C)$ would require linear memory complexity in the number of context tokens, i.e., $\mathcal{O}(|\mathcal{D}_C||L|)$. Instead, we propose to leverage the aforementioned update operation and split the input data $\mathcal{D}_C$ into $|\mathcal{D}_C|/b_C$ batches of input data points of size up to b_C (a pre-specified constant), i.e., $\mathcal{D}_C = \cup_{i=1}^{|\mathcal{D}_C|/b_C} \mathcal{D}_C^{(i)}$. Computing $\text{CrossAttention}(L, \mathcal{D}_C)$ is then equivalent to performing an update $|\mathcal{D}_C|/b_C - 1$ times:

$$\text{CA}(L, \mathcal{D}_C) = \text{UPDATE}(\mathcal{D}_C^{(1)}, \text{UPDATE}(\mathcal{D}_C^{(2)}, \dots \text{UPDATE}(\mathcal{D}_C^{(|\mathcal{D}_C|/b_C-1)}, \text{CA}(L_B, \mathcal{D}_C^{(|\mathcal{D}_C|/b_C)}))))$$

Computing $\text{CrossAttention}(L, \mathcal{D}_C^{(|\mathcal{D}_C|/b_C)})$ requires $\mathcal{O}(b_C|L|)$ memory. After its computation, the memory can be freed up, so that each of the subsequent UPDATE operations can re-use the memory space. Each of the update operations also only uses $\mathcal{O}(b_C|L|)$ constant memory. As such, $\text{CrossAttention}(L, \mathcal{D}_C)$ only needs $\mathcal{O}(b_C|L|) = \mathcal{O}(|L|)$ memory in total, i.e., a memory amount that is independent of the number of context data points. Notably, b_C is a hyperparameter constant which trades off the memory and time complexity.

3.2. Constant Memory Attention Block (CMAB)

Efficiently computing $\text{CrossAttention}(L, \mathcal{D}_C \cup \mathcal{D}_U)$ from $\text{CrossAttention}(L, \mathcal{D}_C)$ require (1) the query tokens remain the same and (2) the new tokens being appended to the old context tokens. However, these requirements do not hold for prior attention models (Vaswani et al., 2017; Lee et al., 2019; Jaegle et al., 2021b). When these attention blocks are stacked, the query tokens of later attention blocks end up being conditioned on the context dataset[1]. As such, when the context dataset changes, the query tokens of these later attention blocks change as well, meaning that the efficient updates property of Cross Attention cannot be applied to existing stacked attention models such as Perceiver (Jaegle et al., 2021a;b). As such, in this section, we specially design the Constant Memory Attention Block (CMAB) that is capable of leveraging the efficiency properties of cross attention while being stacked[2].

The Constant Memory Attention Block (Figure 1) takes as input a fixed-sized set of latent vectors L_I and the context data $\mathcal{D}_C$ and outputs a new set of latent vectors L'_I. Within each CMAB is a unique fixed-sized set of latents L_B learned during training. When stacking CMABs, the output latent vectors of the previous CMAB are fed as the input latent vectors to the next, i.e., $L_I \leftarrow L'_I$. The value of L_I of the first stacked CMAB block is learned.

In the first phase, CMAB applies cross attention between the context data $\mathcal{D}_C$ and the block-wise latent vectors L_B, compressing the data representation. Crucially, since L_B is a set of fixed learned latent vectors unique to each CMAB block, the query tokens are fixed as required regardless of stacking; as a result, this cross attention possesses the efficiency properties of Cross Attention. Afterwards, a self attention is applied to compute higher-order information:

$$L'_B = \text{SelfAttention}(\text{CrossAttention}(L_B, \mathcal{D}_C))$$

The second phase is designed to make CMABs stackable, i.e., computing L'_I from L_I. Specifically, cross attention

[1] Further details are included in Appendix A.4.

[2] See Appendix B for insights regarding the construction of CMAB.

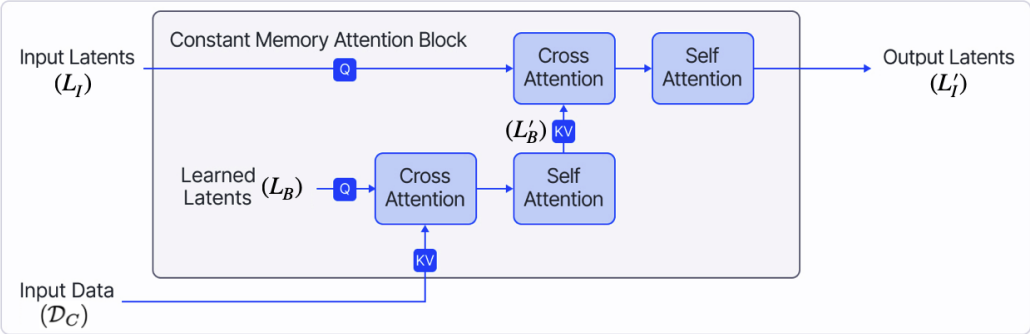


Figure 1. Constant Memory Attention Block (CMAB). CMABs are stackable attention blocks that (i) are permutation invariant in the input data, (ii) compute their output in constant memory, and (iii) compute updates in constant computation per new data point. Notably, L_B is a learned set of latents unique to the block, allowing CMABs to take advantage of the efficient update property of Cross Attention.

is performed by taking in as query tokens the prior latents L_I and as context the compressed data representation L'_B. Afterwards, an additional self-attention is used to compute further higher-order information, resulting in the output vectors L'_I:

$$L'_I = \text{SelfAttention}(\text{CrossAttention}(L_I, L'_B))$$

In terms of the computational complexity of CMAB, the first phase has an overall complexity of $\mathcal{O}(|\mathcal{D}_C||L_B| + |L_B|^2)$ and the second phase has an overall complexity of $\mathcal{O}(|L_B||L_I| + |L_I|^2)$. Since the number of latents $|L_B|$ and $|L_I|$ are hyperparameter constants, therefore these complexities are all constant except for the $\mathcal{O}(|\mathcal{D}_C||L_B|)$ factor induced by the first phase's $\text{CrossAttention}(L_B, \mathcal{D}_C)$. However, due to L_B being fixed, CMABs can leverage the efficiency properties of Cross Attention, resulting in the following CMAB-specific properties: (1) perform updates in constant computation and (2) compute their output in constant memory. In contrast, methods like Transformer or Perceiver require linear or quadratic memory and re-computing from scratch when given new context data. Furthermore, since Cross Attention is permutation invariant in the context, CMABs are also (3) permutation invariant by default.

3.3. Constant Memory Attentive Neural Processes (CMANPs)

In this section, we introduce Constant Memory Attentive Neural Processes (CMANPs), a memory efficient variant of Neural Processes (Figure 2) that leverages the stacked CMAB blocks for efficiency. The conditioning, querying, and updating phases in CMANPs work as follows:

Conditioning Phase: In the conditioning phase, the CMAB blocks encode the context dataset into a set of latent vectors L_i. The first block takes as input a set of meta-learned latent vectors L_0 (i.e., L_I in CMABs) and the context dataset $\mathcal{D}_C$, and outputs a set of encodings L_1 (i.e., L'_I in CMABs). The output latents of each block are passed as the input latents to the next CMAB block.

$$L_i = \textbf{CMAB}(L_{i-1}, \mathcal{D}_C)$$

Since CMABs can compute their output in constant memory, CMANPs can also perform this conditioning phase in constant memory.

Querying Phase: In the querying phase, the deployed model retrieves information from the fixed size outputs of the CMAB blocks (L_i) to make predictions for the query data points (X_{query}). Beginning with $X^0_{query} \leftarrow X_{query}$, when making a prediction for the query data points X_{query}, information is retrieved via cross-attention.

$$X^i_{query} = \text{CrossAttention}(X^{i-1}_{query}, L_i)$$

Updating Phase: In the updating phase, the NP receives a batch of new data points $\mathcal{D}_U$ to include in the context dataset. CMANPs leverage the efficient update mechanism of CMABs to perform this phase efficiently (i.e., constant computation per new data point). Beginning with $L_0^{updated} \leftarrow L_0$, the CMAB blocks are updated sequentially using the updated output of the previous CMAB block as follows:

$$L_i^{updated} = \textbf{CMAB}(L_{i-1}^{updated}, \mathcal{D}_C \cup \mathcal{D}_U)$$

Since CMABs perform updates in constant memory irrespective of the number of context data points, CMANPs also perform updates in constant computation and memory. In Table 1 and 2, we compare the memory complexities of state-of-the-art Neural Processes, showing the efficiency gains of CMANPs.

3.3.1. Autoregressive Not-Diagonal Extension

In many settings where NPs are applied such as Image Completion, the target data points are correlated and their

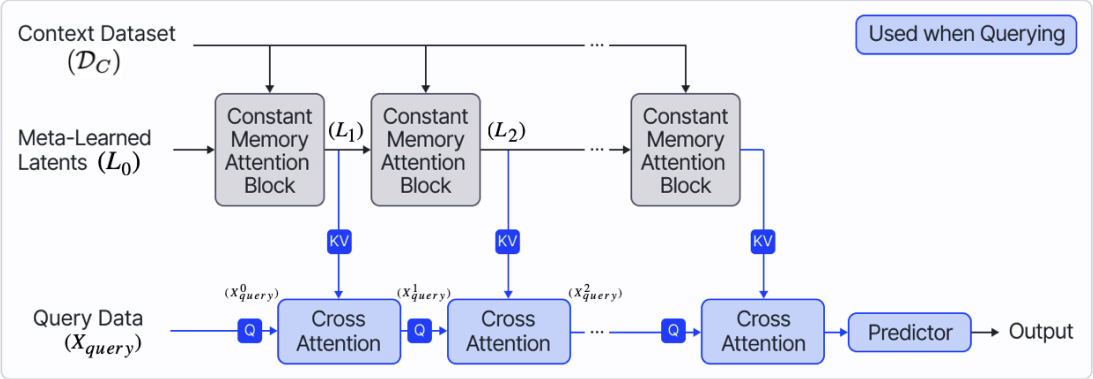


Figure 2. Constant Memory Attentive Neural Processes (CMANPs). CMANPs perform the conditioning, querying, and updating phases in constant memory with respect to the size of the context dataset $|\mathcal{D}_C|$.

	Memory Complexity				
	Cond.	**Query**		**Update**	
In Terms of	$\lvert\mathcal{D}_C\rvert$	$\lvert\mathcal{D}_C\rvert$	M	$\lvert\mathcal{D}_C\rvert$	$\lvert\mathcal{D}_U\rvert$
TNP-D	N/A	✗	✗	N/A	N/A
LBANP	✓	✓	✓	✓	✓
CMANP (Ours)	✓	✓	✓	✓	✓

Table 1. Comparison of Memory Complexities of top-performing Neural Processes with respect to the number of context data points $|\mathcal{D}_C|$, number of target data points in a batch M, and the number of new data points in an update $|\mathcal{D}_U|$. (Green) Checkmarks represent requiring constant memory, (Orange) half checkmarks represent requiring linear memory, and (Red) Xs represent requiring quadratic or more memory. A larger table with all baselines is included in the Appendix (Table 8).

	Memory Complexity				
	Cond.	**Query**		**Update**	
In Terms of	$\lvert\mathcal{D}_C\rvert$	$\lvert\mathcal{D}_C\rvert$	M	$\lvert\mathcal{D}_C\rvert$	$\lvert\mathcal{D}_U\rvert$
TNP-ND	N/A	✗	✗	N/A	N/A
LBANP-ND	✓	✓	✗	✓	✓
CMANP-AND	✓	✓	✓	✓	✓

Table 2. Comparison of Memory Complexities of Not-Diagonal extensions of Neural Processes.

predictive distribution are evaluated altogether. As such, prior works (Nguyen & Grover, 2022; Feng et al., 2023) have proposed Not-Diagonal extensions of NPs which predicts the mean and a full covariance matrix, typically via a low-rank approximation. This is in contrast to the vanilla (Diagonal) variants which predict the mean and a diagonal covariance matrix. Not-Diagonal methods, however, are not scalable, requiring quadratic memory in the number of target data points due to outputting a full covariance matrix.

Leveraging the efficient updates property of CMABs, we propose CMANP-AND (Autoregressive Not-Diagonal). During training, CMANP-AND follows the framework of prior Not-Diagonal variants. When deployed, the model is treated as an autoregressive model that makes predictions in blocks of size b_Q data points. For each block prediction, a mean and full covariance matrix is computed via a low-rank approximation. Sampled predictions of prior blocks are used to make predictions for later blocks. The first block is sampled as follows: $\hat{Y}_1 \sim \mathcal{N}(\mu_\theta(\mathcal{D}_C^0, X_1), \Sigma_\theta(\mathcal{D}_C^0, X_1))$ where $\hat{Y}_1 = \hat{y}_{N+1:N+b_Q}$ and $X_1 = x_{N+1:N+b_Q}$. Afterwards, by leveraging the efficient update mechanism, CMANP-AND performs an update using the sampled predictions $\{(x_i, \hat{y}_i)\}_{N+1}^{N+b_Q}$ as new context data points, meaning that CMANP-AND is now conditioned on a new context dataset $\mathcal{D}_C^1$ where $\mathcal{D}_C^1 = \mathcal{D}_C^0 \cup \{(x_i, \hat{y}_i)\}_{N+1}^{N+b_Q}$. The general formulation is as follows: $\hat{Y}_{k+1} \sim \mathcal{N}(\mu_\theta(\mathcal{D}_C^k, X_{k+1}), \Sigma_\theta(\mathcal{D}_C^k, X_{k+1}))$ where k is the number of blocks already processed, $\hat{Y}_{k+1} = \hat{y}_{N+kb_Q+1:N+(k+1)b_Q}$, $X_{k+1} = x_{N+kb_Q+1:N+(k+1)b_Q}$, and $\mathcal{D}_C^k = \{(x_i, y_i)\}_{i=1}^N \cup \{(x_i, \hat{y}_i)\}_{N+1}^{N+kb_Q}$ is the context dataset. The hyperparameter b_Q controls (1) the computational cost of the model in terms of memory and sequential computation length and (2) the performance of the model. Lower values of b_Q allow for modelling more complex distributions, offering better performance but requiring more forward passes of the model. Since b_Q is a constant, this Autoregressive Not-Diagonal extension makes predictions significantly more efficiently than the prior Not-Diagonal variants which were quadratic in memory. As such, CMANP-AND can scale to a larger number of data points than prior methods (LBANP-ND and TNP-ND). Big-$\mathcal{O}$ complexity analysis is included in Appendix A.3.

4. Experiments

In this section, we aim to evaluate the empirical performance of CMANPs and provide an analysis of CMANPs.

To do so, we compare CMANPs against a large variety of members of the Neural Process family on standard NP benchmarks: image completion and meta-regression. Specifically, we compare against Conditional Neural Processes (CNPs) (Garnelo et al., 2018a), Neural Processes (NPs) (Garnelo et al., 2018b), Bootstrapping Neural Processes (BNPs) (Lee et al., 2020), (Conditional) Attentive Neural Processes (C)ANPs (Kim et al., 2019), Bootstrapping Attentive Neural Processes (BANPs) (Lee et al., 2020), Latent Bottlenecked Attentive Neural Processes (LBANPs) (Feng et al., 2023), and Transformer Neural Processes (TNPs) (Nguyen & Grover, 2022). We also compare against the Not-Diagonal extensions of the state-of-the-art methods (i.e., LBANP-ND and TNP-ND).

For consistency, we set the number of latents (i.e., bottleneck size) $|L_I| = |L_B| = 128$ across all experiments. We also set $b_Q = 5$. To fairly compare iterative attention and CMABs, we report results for LBANPs with the same sized bottleneck (i.e., number of latents $L = 128$) as CMANPs across all experiments. Due to space limitations, additional hyperparameters and experiments are included in the appendix[3].

4.1. Image Completion

In these experiments, we consider the image completion setting. The model is given a set of pixel values of an image and aims to predict the remaining pixels of the image. Each image corresponds to a unique function (Garnelo et al., 2018b). In these experiments, the x values are rescaled to [-1, 1] and the y values are rescaled to $[-0.5, 0.5]$. For each task, a randomly selected set of pixels are selected as context data points and target data points.

EMNIST (Cohen et al., 2017) comprises black and white images of handwritten letters of 32×32 resolution. 10 classes are used for training. The context and target data points are sampled according to $N \sim \mathcal{U}[3, 197)$ and $M \sim \mathcal{U}[3, 200 - N)$ respectively. CelebA (Liu et al., 2015) comprises coloured images of celebrity faces. Methods are evaluated on various resolutions to show the scalability of the methods. In CelebA32, images are downsampled to 32×32 and the number of context and target data points are sampled according to $N \sim \mathcal{U}[3, 197)$ and $M \sim \mathcal{U}[3, 200 - N)$ respectively. In CelebA64, the images are down-sampled to 64×64 and $N \sim \mathcal{U}[3, 797)$ and $M \sim \mathcal{U}[3, 800 - N)$. In CelebA128, the images are down-sampled to 128×128 and $N \sim \mathcal{U}[3, 1597)$ and $M \sim \mathcal{U}[3, 1600 - N)$.

Results. All NP baselines (see Table 3) were able to be evaluated on CelebA (32 x 32) and EMNIST. However, many baselines were not able to be trained on CelebA (64 x 64) and CelebA (128 x 128) in the available GPU memory due to requiring a quadratic amount of memory including all prior Not-Diagonal extensions of NPs. In contrast, CMANP(-AND) was not affected by this limitation due to being significantly more memory efficient than prior Not-Diagonal variants. The results show that CMANP-AND achieves clear state-of-the-art results on CelebA (32x32), CelebA (64x64), and CelebA (128x128) while being scalable to more data points than prior Not-Diagonal variants. Furthermore, CMANP-AND achieves results competitive with state-of-the-art EMNIST.

Empirical Computational Complexity Results. In Figure 3, we analyze the empirical memory and runtime used by CMANPs compared with various state-of-the-art Neural Process models. Comparing Not-Diagonal extensions of NPs (Figure 3 (Left)), we see that the memory usage of both TNP-ND and LBANP-ND scale quadratically with respect to the number of target data points, making these methods infeasible to be applied to settings with a larger number of target data points as shown in our CelebA experiments. In contrast, CMANP-AND can scale to a far larger number of target data points. Comparing the vanilla variants of NPs (Figure 3 (Middle)), we see that TNP-D (transformer-based model) scales quadratically with respect to the number of context data points while LBANP (Perceiver-based model) scales linearly. In contrast, CMANP (CMAB-based model) only requires a low constant amount of memory regardless of the number of context data points. As a result, we can see that CMANPs use a significantly less amount of memory, allowing them to scale to more data points while achieving results competitive with state-of-the-art.

In terms of runtime (Figure 3 (Right)), LBANPs and CMANPs are comparable as they both only require retrieving information from a fixed-size set of latents. In contrast, TNPs are quadratic due to leveraging transformers during queries. Due to space limitations, we include in the appendix the runtime complexity plot of the update operation which shows that CMANPs' efficient update mechanism is significantly more efficient than that of prior state-of-the-art.

4.2. Meta-Regression

In this experiment, the goal is to model an unknown function f and make predictions for a batch of M target data points given a batch of N context data points. During each training epoch, a batch of $B = 16$ functions are sampled from a GP prior with an RBF kernel $f_i \sim GP(m, k)$ where $m(x) = 0$ and $k(x, x') = \sigma_f^2 \exp(\frac{-(x-x')^2}{2l^2})$. The hyperparameters are sampled according to $l \sim \mathcal{U}[0.6, 1.0)$, $\sigma_f \sim \mathcal{U}[0.1, 1.0)$, $N \sim \mathcal{U}[3, 47)$, and $M \sim \mathcal{U}[3, 50 - N)$. After training, the models are evaluated according to the log-likelihood of the targets on functions sampled from GPs with RBF and Matern $5/2$ kernels.

[3]Code: https://github.com/BorealisAI/constant-memory-anp.

Method	CelebA			EMNIST	
	32x32	64x64	128x128	Seen (0-9)	Unseen (10-46)
CNP	2.15 ± 0.01	2.43 ± 0.00	2.55 ± 0.02	0.73 ± 0.00	0.49 ± 0.01
CANP	2.66 ± 0.01	3.15 ± 0.00	—	0.94 ± 0.01	0.82 ± 0.01
NP	2.48 ± 0.02	2.60 ± 0.01	2.67 ± 0.01	0.79 ± 0.01	0.59 ± 0.01
ANP	2.90 ± 0.00	—	—	0.98 ± 0.00	0.89 ± 0.00
BNP	2.76 ± 0.01	2.97 ± 0.00	—	0.88 ± 0.01	0.73 ± 0.01
BANP	3.09 ± 0.00	—	—	1.01 ± 0.00	0.94 ± 0.00
TNP-D	3.89 ± 0.01	5.41 ± 0.01	—	**1.46 ± 0.01**	**1.31 ± 0.00**
LBANP	3.97 ± 0.02	5.09 ± 0.02	5.84 ± 0.01	1.39 ± 0.01	1.17 ± 0.01
CMANP (Ours)	3.93 ± 0.05	5.02 ± 0.14	5.55 ± 0.01	1.36 ± 0.01	1.09 ± 0.01
TNP-ND	5.48 ± 0.02	—	—	**1.50 ± 0.00**	**1.31 ± 0.00**
LBANP-ND	5.57 ± 0.03	—	—	1.42 ± 0.01	1.14 ± 0.01
CMANP-AND (Ours)	**6.31 ± 0.04**	**6.96 ± 0.07**	**7.15 ± 0.14**	**1.48 ± 0.03**	1.19 ± 0.03

Table 3. Image Completion Experiments. Each method is evaluated with 5 different seeds according to the log-likelihood (higher is better). The "dash" represents methods that could not be run because of the large memory requirement.

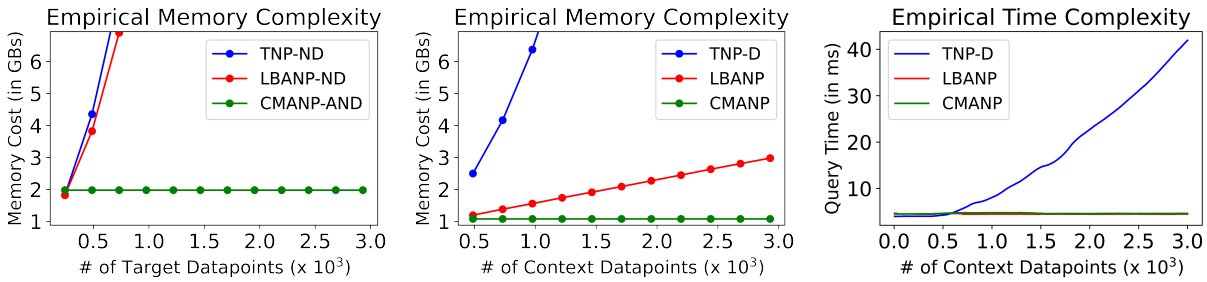


Figure 3. Computational Complexity Comparison Plots. (Left) Empirical memory usage comparison of CMANP-AND with state-of-the-art Not-Diagonal NPs. (Middle) Empirical memory usage comparison of CMANPs with state-of-the-art NPs. (Right) Empirical runtime comparison of CMANPs with state-of-the-art NPs.

Results. As shown in Table 4, CMANP-AND achieves comparable results to TNP-ND and outperforms all other baselines by a significant margin while only requiring constant memory. Furthermore, we see that the vanilla version of CMANP (CMAB-based model) and LBANP (Perceiver-based model (Jaegle et al., 2021b)) achieve similar performance, showing that CMANPs achieve competitive performance while being significantly more efficient (constant memory and constant computation updates).

4.2.1. Analysis

Effect of Block Size (b_Q): In Figure 4 (Left and Middle), we evaluate the test-time performance and runtime for CMANP-AND with respect to the block size (b_Q). The smaller the block size, the better the performance. This is expected as the autoregressive nature of CMANP-AND allows for more flexible predictive distribution and hence better performance. This, however, comes at the cost of an increased time complexity. In conjunction, these plots show that there is a trade-off between the time cost and performance. As such, b_Q is a hyperparameter that can be adjusted based on

Method	RBF	Matern 5/2
CNP	0.26 ± 0.02	0.04 ± 0.02
CANP	0.79 ± 0.00	0.62 ± 0.00
NP	0.27 ± 0.01	0.07 ± 0.01
ANP	0.81 ± 0.00	0.63 ± 0.00
BNP	0.38 ± 0.02	0.18 ± 0.02
BANP	0.82 ± 0.01	0.66 ± 0.00
TNP-D	1.39 ± 0.00	0.95 ± 0.01
LBANP	1.27 ± 0.02	0.85 ± 0.02
CMANP (Ours)	1.24 ± 0.01	0.80 ± 0.01
TNP-ND	**1.46 ± 0.00**	**1.02 ± 0.00**
LBANP-ND	1.24 ± 0.03	0.78 ± 0.02
CMANP-AND (Ours)	**1.48 ± 0.03**	0.96 ± 0.01

Table 4. 1-D Meta-Regression Experiments with log-likelihood metric (higher is better).

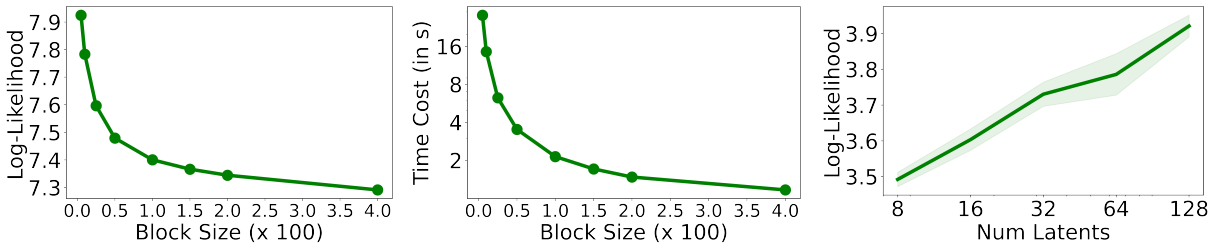


Figure 4. Analyses Plots. (Left) CMANP's runtime relative to the predictive block size (b_Q). (Middle) CMANP's performance relative to the predictive block size (b_Q). (Right) CMANP's performance relative to the number of latent vectors ($|L_I| = |L_B|$).

the available resources.

Varying Number of Latents: Figure 4 (Right) evaluates the result of varying the number of latents ($|L_I| = |L_B|$), showing that increasing the number of latents (i.e., the size of the bottleneck) improves the performance of the model.

Extending CMAB beyond Neural Processes: Constant Memory Attention Blocks (1) do not leverage any modality-specific components and (2) are permutation invariant by default like Transformers. As such, CMABs can naturally extend beyond that of Neural Processes. As a proof of concept, we experiment with extending CMABs to next-event prediction (Temporal Point Processes (TPPs)) by replacing the transformer in Transformer Hawkes Process (THP) (Zuo et al., 2020), a popular method in the TPP literature, with our proposed Constant Memory Attention Block, resulting in the Constant Memory Hawkes Process. In Table 5, we see that CMHP achieves comparable results with that of THP on a popular dataset in the TPP literature: Reddit. Crucially, CMHP only requires constant memory unlike the quadratic memory required by that of THP. Furthermore, CMHP efficiently updates its model with new data as it arrives over time which is typical in event sequence data, making it significantly more efficient than the quadratic computation required by THP.

Model	Reddit		
	RMSE	NLL	ACC
THP	**0.238 ± 0.028**	**0.268 ± 0.098**	**0.610 ± 0.002**
CMHP	**0.262 ± 0.037**	**0.528 ± 0.209**	**0.609 ± 0.003**

Table 5. Temporal Point Processes Experiments.

5. Related Work

Meta-learning has two learning phases (inner/task and outer/meta). In this work, we consider Neural Processes (NPs), a member of the model-based (or black-box) meta-learning family. Model-based/black-box methods' perform their inner learning phase via a forward pass through a model, often a neural network. Several architectures have been proposed for the inner learning phase such as convolutional networks (Mishra et al., 2018), recurrent networks (Zintgraf et al., 2021), attention (Feng et al., 2023), and hypernetworks (Chen & Wang, 2022). For an in-depth overview of the meta-learning literature, we refer the reader to the following meta-learning survey (Hospedales et al., 2022).

NPs have been applied to a wide range of applications which include sequence data (Singh et al., 2019; Willi et al., 2019), modelling stochastic physics fields (Holderrieth et al., 2021), robotics (Chen et al., 2022; Li et al., 2022b), event prediction (Bae et al., 2023), and climate modeling (Vaughan et al., 2021). In doing so, there have been several methods proposed for encoding the context dataset. For example, CNPs (Garnelo et al., 2018a) encode the context set via a deep sets encoder (Zaheer et al., 2017), NPs (Garnelo et al., 2018b) propose to encode functional stochasticity via a latent variable. ConvCNPs (Gordon et al., 2019) use convolutions to build in translational equivariance. ANPs (Kim et al., 2019), LBANPs (Feng et al., 2023), and TNPs (Nguyen & Grover, 2022) use various kinds of attention. Recent work (Bruinsma et al., 2023) builds on CNPs and ConvCNPs by proposing to make them autoregressive at deployment. For an in-depth overview of NPs, we refer the reader to the recent survey work (Jha et al., 2022).

Transformers (Vaswani et al., 2017) have achieved a large amount of success in a wide range of applications. However, the quadratic scaling of Transformers limits their applications. As such, there have been many follow-up works on efficient variants. However, very few works have achieved constant memory complexity. Rabe & Staats (2022) showed that self-attention can be computed in constant memory at the expense of an overall quadratic computation. In contrast, CMABs only require linear computation and constant memory, making it significantly more efficient. Wu et al. (2022) and Peng et al. (2023) proposed Memformer and RWKV respectively, constant memory versions of transformer for sequence modelling problems. However, these methods are dependent on the order of the tokens, limiting their applications. In contrast, CMABs are by default permutation-invariant, allowing them to be more flexible

in applications with their data modality. For an in-depth overview of follow-up works to Transformers, we refer the reader to the recent survey works (Khan et al., 2022; Lin et al., 2022).

Although CMABs have an efficient update mechanism reminiscent of RNNs (Cho et al., 2014; Chung et al., 2014; Hochreiter & Schmidhuber, 1997), their applications are different. RNNs are sensitive to input order, making their ideal setting applications which use sequential data. In contrast, by design, CMABs are by default permutation-invariant. Due to their long computation graph, RNNs also have issues such as vanishing gradients, making training these models with a large number of data points difficult. CMABs do not have issues with vanishing gradients since their ability to update efficiently is a fixed property of the module rather than RNN's learned mechanism.

6. Conclusion

In this work, we showed that Cross Attention can be updated efficiently with new context data and computed in a constant amount of memory irrespective of the number of context data points. Leveraging these properties, we introduced CMAB (Constant Memory Attention Block), a novel attention block that (1) is permutation invariant, (2) computes its output in constant memory, and (3) performs updates in constant computation. Building on CMAB, we proposed Constant Memory Attentive Neural Processes (CMANPs), a new memory-efficient NP variant. Leveraging the efficient updates property of CMAB, we introduced CMANP-AND (Autoregressive Not-Diagonal extension). Empirically, we show that CMANP(-AND) achieves state-of-the-art results while being significantly more efficient than prior state-of-the-art methods. In our analysis, we showed that increasing the size of the latent bottleneck can improve CMANPs' performance. Lastly, we showed a proof of concept extending CMABs beyond Neural Processes.

Impact Statement

In this paper, we improve the memory efficiency of Neural Processes, a family of deep learning models for uncertainty estimation. In order to deploy deep learning models to real-world settings, it is imperative that the models be memory efficient whilst simultaneously being able to capture their predictive uncertainty; incorrect or overconfident predictions can have dire consequences in mission- or safety-critical settings (Tambon et al., 2022). Furthermore, designing memory efficient deep learning models is important for a variety of reasons, for example: (1) With the growing popularity of low-memory/compute domains (e.g., IoT devices and mobile phones), it is crucial that deep learning models be designed memory efficiently in order to be deployable on the device. (2) In addition, memory access is energy intensive (Li et al., 2022a). Since many safety-critical settings (e.g., autonomous vehicles and mobile robots in disaster response) have a limited energy supply, it is therefore crucial to develop memory efficient models that can simultaneously model predictive uncertainty such as Neural Processes.

References

Bae, W., Ahmed, M. O., Tung, F., and Oliveira, G. L. Meta temporal point processes. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=QZfdDpTX1uM.

Bruinsma, W., Markou, S., Requeima, J., Foong, A. Y. K., Andersson, T., Vaughan, A., Buonomo, A., Hosking, S., and Turner, R. E. Autoregressive conditional neural processes. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=OAsXFPBfTBh.

Chen, R., Gao, N., Vien, N. A., Ziesche, H., and Neumann, G. Meta-learning regrasping strategies for physical-agnostic objects. *arXiv preprint arXiv:2205.11110*, 2022.

Chen, Y. and Wang, X. Transformers as meta-learners for implicit neural representations. In *European Conference on Computer Vision*, pp. 170–187. Springer, 2022.

Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Cohen, G., Afshar, S., Tapson, J., and Van Schaik, A. Emnist: Extending mnist to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)*, pp. 2921–2926. IEEE, 2017.

Feng, L., Hajimirsadeghi, H., Bengio, Y., and Ahmed, M. O. Latent bottlenecked attentive neural processes. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=yIxtevizEA.

Garnelo, M., Rosenbaum, D., Maddison, C., Ramalho, T., Saxton, D., Shanahan, M., Teh, Y. W., Rezende, D., and Eslami, S. A. Conditional neural processes. In *International Conference on Machine Learning*, pp. 1704–1713. PMLR, 2018a.

Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S., and Teh, Y. W. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018b.

Gordon, J., Bruinsma, W. P., Foong, A. Y., Requeima, J., Dubois, Y., and Turner, R. E. Convolutional conditional neural processes. In *International Conference on Learning Representations*, 2019.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

Holderrieth, P., Hutchinson, M. J., and Teh, Y. W. Equivariant learning of stochastic fields: Gaussian processes and steerable conditional neural processes. In *International Conference on Machine Learning*, pp. 4297–4307. PMLR, 2021.

Hospedales, T., Antoniou, A., Micaelli, P., and Storkey, A. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):5149–5169, 2022.

Jaegle, A., Borgeaud, S., Alayrac, J.-B., Doersch, C., Ionescu, C., Ding, D., Koppula, S., Zoran, D., Brock, A., Shelhamer, E., et al. Perceiver io: A general architecture for structured inputs & outputs. In *International Conference on Learning Representations*, 2021a.

Jaegle, A., Gimeno, F., Brock, A., Vinyals, O., Zisserman, A., and Carreira, J. Perceiver: General perception with iterative attention. In *International conference on machine learning*, pp. 4651–4664. PMLR, 2021b.

Jha, S., Gong, D., Wang, X., Turner, R. E., and Yao, L. The neural process family: Survey, applications and perspectives. *arXiv preprint arXiv:2209.00517*, 2022.

Khan, S., Naseer, M., Hayat, M., Zamir, S. W., Khan, F. S., and Shah, M. Transformers in vision: A survey. *ACM computing surveys (CSUR)*, 54(10s):1–41, 2022.

Kim, H., Mnih, A., Schwarz, J., Garnelo, M., Eslami, A., Rosenbaum, D., Vinyals, O., and Teh, Y. W. Attentive neural processes. In *International Conference on Learning Representations*, 2019.

Kumar, S., Zhang, X., and Leskovec, J. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1269–1278. ACM, 2019.

Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., and Teh, Y. W. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pp. 3744–3753. PMLR, 2019.

Lee, J., Lee, Y., Kim, J., Yang, E., Hwang, S. J., and Teh, Y. W. Bootstrapping neural processes. *Advances in neural information processing systems*, 33:6606–6615, 2020.

Li, P. Z. X., Karaman, S., and Sze, V. Memory-efficient gaussian fitting for depth images in real time. In *2022 International Conference on Robotics and Automation (ICRA)*, pp. 8003–8009, 2022a. doi: 10.1109/ICRA46639.2022.9811682.

Li, Y., Gao, N., Ziesche, H., and Neumann, G. Category-agnostic 6d pose estimation with conditional neural processes. *arXiv preprint arXiv:2206.07162*, 2022b.

Liang, H. and Gao, J. How neural processes improve graph link prediction. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3543–3547. IEEE, 2022.

Lin, T., Wang, Y., Liu, X., and Qiu, X. A survey of transformers. *AI Open*, 2022.

Lin, X., Wu, J., Zhou, C., Pan, S., Cao, Y., and Wang, B. Task-adaptive neural process for user cold-start recommendation. In *Proceedings of the Web Conference 2021*, pp. 1306–1316, 2021.

Liu, Z., Luo, P., Wang, X., and Tang, X. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. A simple neural attentive meta-learner. In *International Conference on Learning Representations*, 2018.

Nguyen, T. and Grover, A. Transformer neural processes: Uncertainty-aware meta learning via sequence modeling. In *International Conference on Machine Learning*, pp. 16569–16594. PMLR, 2022.

Peng, B., Alcaide, E., Anthony, Q. G., Albalak, A., Arcadinho, S., Biderman, S., Cao, H., Cheng, X., Chung, M. N., Derczynski, L., Du, X., Grella, M., GV, K. K., He, X., Hou, H., Kazienko, P., Kocon, J., Kong, J., Koptyra, B., Lau, H., Lin, J., Mantri, K. S. I., Mom, F., Saito, A., Song, G., Tang, X., Wind, J. S., Woźniak, S., Zhang, Z., Zhou, Q., Zhu, J., and Zhu, R.-J. RWKV: Reinventing RNNs for the transformer era. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=7SaXczaBpG.

Rabe, M. N. and Staats, C. Self-attention does not need $o(n^2)$ memory. *arXiv preprint arXiv:2112.05682*, 2022.

Requeima, J., Gordon, J., Bronskill, J., Nowozin, S., and Turner, R. E. Fast and flexible multi-task classification using conditional neural adaptive processes. *Advances in Neural Information Processing Systems*, 32, 2019.

Riquelme, C., Tucker, G., and Snoek, J. Deep bayesian bandits showdown: An empirical comparison of bayesian

deep networks for thompson sampling. *arXiv preprint arXiv:1802.09127*, 2018.

Shchur, O., Biloš, M., and Günnemann, S. Intensity-free learning of temporal point processes. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HygOjhEYDH.

Singh, G., Yoon, J., Son, Y., and Ahn, S. Sequential neural processes. *Advances in Neural Information Processing Systems*, 32, 2019.

Tambon, F., Laberge, G., An, L., Nikanjam, A., Mindom, P. S. N., Pequignot, Y., Khomh, F., Antoniol, G., Merlo, E., and Laviolette, F. How to certify machine learning based safety-critical systems? a systematic literature review. *Automated Software Engineering*, 29(2):38, 2022.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Vaughan, A., Tebbutt, W., Hosking, J. S., and Turner, R. E. Convolutional conditional neural processes for local climate downscaling. *arXiv preprint arXiv:2101.07950*, 2021.

Willi, T., Masci, J., Schmidhuber, J., and Osendorfer, C. Recurrent neural processes. *arXiv preprint arXiv:1906.05915*, 2019.

Wu, Q., Lan, Z., Qian, K., Gu, J., Geramifard, A., and Yu, Z. Memformer: A memory-augmented transformer for sequence modeling. In *Findings of the Association for Computational Linguistics: AACL-IJCNLP 2022*, pp. 308–318, 2022.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Zintgraf, L., Schulze, S., Lu, C., Feng, L., Igl, M., Shiarlis, K., Gal, Y., Hofmann, K., and Whiteson, S. Varibad: Variational bayes-adaptive deep rl via meta-learning. *Journal of Machine Learning Research*, 22(289):1–39, 2021.

Zuo, S., Jiang, H., Li, Z., Zhao, T., and Zha, H. Transformer hawkes process. In *International conference on machine learning*, pp. 11692–11702. PMLR, 2020.

A. Appendix: Additional Proof Details

In this section, we (1) provide formal proof for Cross Attention's efficient updates property, (2) derive a practical implementation of the efficient updates property that avoids numerical issues, (3) show that CMANPs satisfy context and target invariance properties, and (4) include computational complexity analysis for CMANP-AND.

A.1. Cross Attention's Efficient Updates Proof

Here, we detail how $\mathrm{CrossAttention}(L, \mathcal{D}_C \cup \mathcal{D}_U)$ can be efficiently computed given $\mathrm{CrossAttention}(L, \mathcal{D}_C)$ in only $\mathcal{O}(|\mathcal{D}_U||L|)$ computation, i.e., an amount of memory independent of the number of original context datapoints $|\mathcal{D}_C|$. For brevity, let $\mathrm{CA} = \mathrm{CrossAttention}(L, \mathcal{D}_C)$ and $\mathrm{CA}' = \mathrm{CrossAttention}(L, \mathcal{D}_C \cup \mathcal{D}_U)$.

Recall, Cross Attention is computed as follows:

$$\mathrm{CrossAttention}(L, \mathcal{D}_C) = \mathrm{softmax}(QK^T)V$$

where $Q = LW_q$ is the query matrix, $K = \mathcal{D}_C W_k$ is the key matrix, and $V = \mathcal{D}_C W_v$ is the value matrix. $W_q, W_k, W_v \in \mathbb{R}^{d \times d}$ are weight matrices (learned parameters) that project the input tokens. $\mathrm{softmax}(QK^T)$ aims to compute the weight to give the respective context tokens for computing a weighted average.

Without loss of generality, for simplicity, we first consider the j-th output vector of the Cross Attention (CA_j). The value of the j-th output of the Cross Attention is as follows:

$$\mathrm{CA}_j = \sum_{i=1}^{|\mathcal{D}_C|} \frac{\exp(s_i)}{c_j} v_i$$

where $s_i = Q_{j,:}(K_{i,:})^T$, $v_i = V_{i,:}$, and $c_j = \sum_{i=1}^{|\mathcal{D}_C|} \exp(s_i)$ (a cached normalizing constant).

When new data points $\mathcal{D}_U$ are added to the context in the computation of $\mathrm{CA}' = \mathrm{CrossAttention}(L, \mathcal{D}_C \cup \mathcal{D}_U)$, the key and value matrices are computed as follows: $K' \leftarrow [\mathcal{D}_C, \mathcal{D}_U]W_k$ and $V' \leftarrow [\mathcal{D}_C, \mathcal{D}_U]W_v$ where $[\mathcal{D}_C, \mathcal{D}_U]$ is the stacking of the new data points $\mathcal{D}_U$ on the original context data $\mathcal{D}_C$. We can simplify the computation of the key value matrices as follows: $K' = [\mathcal{D}_C, \mathcal{D}_U]W_k = [\mathcal{D}_C W_k, \mathcal{D}_U W_k] = [K, \mathcal{D}_U W_k]$ and $V' = [\mathcal{D}_C, \mathcal{D}_U]W_v = [V, \mathcal{D}_U W_v]$. As a result, these key value matrices are simply the original key value matrices with $|\mathcal{D}_U|$ additional rows corresponding to that of the new data points. In contrast, the query matrix is unchanged, i.e., $Q' = LW_q = Q$.

The updated Cross Attention ($\mathrm{CA}' = \mathrm{CrossAttention}(L, \mathcal{D}_C \cup \mathcal{D}_U)$) with $\mathcal{D}_U$ new data points is computed as follows:

$$\mathrm{CA}'_j = \sum_{i=1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \frac{\exp(s'_i)}{c'_j} v'_i = \sum_{i=1}^{|\mathcal{D}_C|} \frac{\exp(s'_i)}{c'_j} v'_i + \sum_{i=|\mathcal{D}_C|+1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \frac{\exp(s'_i)}{c'_j} v'_i$$

where $s'_i = Q'_{j,:}(K'_{i,:})^T$, $v'_i = V'_{i,:}$, and $c'_j = \sum_{i=1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \exp(s'_i)$ (a new normalizing constant).

Since $K' = [K, \mathcal{D}_U W_k]$ and $Q' = Q$, thus $\forall i \in \{1, \ldots, |\mathcal{D}_C|\}$ $s'_i = Q'_{j,:}(K'_{i,:})^T = Q_{j,:}(K_{i,:})^T = s_i$. Building on this and the fact that $c_j = \sum_{i=1}^{|\mathcal{D}_C|} \exp(s_i)$, we also have $c'_j = \sum_{i=1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \exp(s'_i) = c_j + \sum_{i=|\mathcal{D}_C|+1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \exp(s'_i)$, i.e., c'_j can be computed from c_j via a rolling sum in $\mathcal{O}(|\mathcal{D}_U|)$.

Furthermore, since $V' = [V, \mathcal{D}_U W_v]$, we also have $\forall i \in \{1, \ldots, |\mathcal{D}_C|\}$ $v'_i = V'_{i,:} = V_{i,:} = v_i$.

Combining the prior derivations, we can re-write the updated $\mathrm{CA'}_j$ as a rolling average that only requires $\mathcal{O}(|\mathcal{D}_U|)$ compute:

$$\begin{aligned}\mathrm{CA'}_j &= \sum_{i=1}^{|\mathcal{D}_C|} \frac{\exp(s_i)}{c'_j} v_i + \sum_{i=|\mathcal{D}_C|+1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \frac{\exp(s'_i)}{c'_j} v'_i \\ &= \frac{c_j}{c'_j} \sum_{i=1}^{|\mathcal{D}_C|} \frac{\exp(s_i)}{c_j} v_i + \sum_{i=|\mathcal{D}_C|+1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \frac{\exp(s'_i)}{c'_j} v'_i \\ &= \frac{c_j}{c'_j} \mathrm{CA}_j + \sum_{i=|\mathcal{D}_C|+1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \frac{\exp(s'_i)}{c'_j} v'_i\end{aligned}$$

where $c'_j = c_j + \sum_{i=|\mathcal{D}_C|+1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \exp(s'_i)$ computed via a $\mathcal{O}(|\mathcal{D}_U|)$ computation rolling sum.

Due to the space limitations of the main paper, we aim to simplify the notation. Note that v_i and s_i were only defined for $i \in \{1, \ldots, |\mathcal{D}_C|\}$. Furthermore, $\forall i \in \{1, \ldots, |\mathcal{D}_C|\}$ $v'_i = v_i$ and $s'_i = s_i$. As such, we abuse the notation and simply replace s'_i and v'_i with s_i and v_i, resulting in the following formulation seen in the main paper:

$$\mathrm{CA'}_j = \frac{c_j}{c'_j} \mathrm{CA}_j + \sum_{i=|\mathcal{D}_C|+1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \frac{\exp(s_i)}{c'_j} v_i$$

There are L queries, so computing the updated output $\mathrm{CA'}$ given CA uses $\mathcal{O}(|\mathcal{D}_U||L|)$ computation in total.

Practical Implementation:

In practice, the aforementioned implementation is not stable due to requiring the computation of c'_j from c_j. Specifically, $c'_j = c_j + \sum_{i=|\mathcal{D}_C|+1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \exp(s_i)$ runs into underflow and overflow problems due to being a sum of exponentials. As such, we derive a different practical formulation to avoid numerical issues.

In the practical formulation, instead of computing $\mathrm{CA'}$ from CA using C' and C, we can instead use $\log(C')$ and $\log(C)$ as follows:

$$\mathrm{CA'_j} = \exp(\log(c_j) - \log(c'_j)) \times \mathrm{CA_j} + \sum_{i=|\mathcal{D}_C|+1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \exp(s_i - \log(c'_j)) v_i$$

$\log(c'_j)$ is computed from $\log(c_j)$ in a rolling sum manner: $\log(c'_j) = \log(c_j) + \mathrm{softplus}(t_j)$ where $t_j = \log(\sum_{i=|\mathcal{D}_C|+1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \exp(s_i - \log(c_j)))$. t_j can be computed accurately using the log-sum-exp trick. This method of implementation avoids the underflow and overflow issue while still computing the same updated Cross Attention result CA'. We detail the derivation of this practical implementation below:

Practical Implementation (Derivation):

$$\begin{aligned}c_j &= \sum_{i=1}^{|\mathcal{D}_C|} \exp(s_i) \\ c'_j &= \sum_{i=1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \exp(s_i) \\ &= \sum_{i=1}^{|\mathcal{D}_C|} \exp(s_i) + \sum_{i=|\mathcal{D}_C|+1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \exp(s_i)\end{aligned}$$

$$\log(c'_j) - \log(c_j) = \log(\sum_{i=1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \exp(s_i)) - \log(\sum_{i=1}^{|\mathcal{D}_C|} \exp(s_i))$$
$$\log(c'_j) = \log(c_j) + \log(\frac{\sum_{i=1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \exp(s_i)}{\sum_{i=1}^{|\mathcal{D}_C|} \exp(s_i)})$$
$$\log(c'_j) = \log(c_j) + \log(1 + \frac{\sum_{i=|\mathcal{D}_C|+1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \exp(s_i)}{\sum_{i=1}^{|\mathcal{D}_C|} \exp(s_i)})$$
$$\log(c'_j) = \log(c_j) + \log(1 + \frac{\sum_{i=|\mathcal{D}_C|+1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \exp(s_i)}{\exp(\log(c_j))})$$
$$\log(c'_j) = \log(c_j) + \log(1 + \sum_{i=|\mathcal{D}_C|+1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \exp(s_i - \log(c_j)))$$

Let $t_j = \log(\sum_{i=|\mathcal{D}_C|+1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \exp(s_i - \log(c_j)))$. Note that t_j can be computed efficiently and accurately using the log-sum-exp trick in $\mathcal{O}(|\mathcal{D}_U|)$. Also, recall the $\mathrm{softplus}$ function is defined as follows: $\mathrm{softplus}(k) = \log(1 + \exp(k))$. Leveraging these, we have the following:

$$\log(c'_j) = \log(c_j) + \log(1 + \exp(t_j))$$
$$= \log(c_j) + \mathrm{softplus}(t_j)$$

Now recall the original update formulation:

$$\mathrm{CA}'_j = \frac{c_j}{c'_j} \times \mathrm{CA}_j \ + \sum_{i=|\mathcal{D}_C|+1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \frac{\exp(s_i)}{c'_j} v_i$$

Re-formulating it using $\log(C)$ and $\log(C')$ instead of C and C' we have the following update formulation:

$$\mathrm{CA}'_j = \exp(\log(c_j) - \log(c'_j)) \times \mathrm{CA}_j + \sum_{i=|\mathcal{D}_C|+1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \exp(s_i - \log(c'_j)) v_i$$

where $\log(c'_j) = \log(c_j) + \log(1 + \sum_{i=|\mathcal{D}_C|+1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \exp(s_i - \log(c_j)))$ and $t_j = \log(\sum_{i=|\mathcal{D}_C|+1}^{|\mathcal{D}_C|+|\mathcal{D}_U|} \exp(s_i - \log(c_j)))$ such that t_j is computed using the log-sum-exp trick. Notably, this formulation avoids the underflow and overflow issues while still only requiring $\mathcal{O}(|\mathcal{D}_U|)$ computation to compute CA'_j.

A.2. Additional Properties

In this section, we show that CMANPs uphold the context and target invariance properties.

Property: Context Invariance. A Neural Process p_θ is context invariant if for any choice of permutation function π, context data points $\{(x_i, y_i)\}_{i=1}^N$, and target data points $x_{N+1:N+M}$,

$$p_\theta(y_{N+1:N+M} | x_{N+1:N+M}, x_{1:N}, y_{1:N}) = p_\theta(y_{N+1:N+M} | x_{N+1:N+M}, x_{\pi(1):\pi(N)}, y_{\pi(1):\pi(N)})$$

Proof Outline: Since CMANPs retrieve information from compressed encodings of the context dataset $\mathcal{D}_C$ computed using CMABs (Constant Memory Attention Blocks). It suffices to show that CMABs compute their output while being order invariant in the context dataset.

In brief CMAB's work as follows:

$$\mathbf{CMAB}(L_I, \mathcal{D}_C) = \mathbf{SA}(\mathbf{CA}(L_I, \mathbf{SA}(\mathbf{CA}(L_B, \mathcal{D}_C))))$$

where L_I is a set of vectors outputted by prior blocks, L_B is a set of vectors whose values are learned during training, and $\mathcal{D}_\mathcal{C}$ are the set of inputs in which we wish to be order invariant in.

The first cross-attention to be computed is $\mathbf{CA}(L_B, \mathcal{D}_C)$. A nice feature of cross-attention is that its order-invariant in the keys and values; in this case, these are embeddings of $\mathcal{D}_C$. In other words, the output of $\mathbf{CA}(L_B, \mathcal{D})$ is order invariant in the input data $\mathcal{D}$.

The inputs to the remaining self-attention and cross-attention blocks take as input a combination of: L_I and the output of $\mathbf{CA}(L_B, \mathcal{D}_C)$, both of which are order invariant in $\mathcal{D}_C$. As such, the output of CMAB is order invariant in $\mathcal{D}_C$. Therefore, CMANPs are also context invariant as required.

Property: Target Equivariance. A model p_θ is target equivariant if for any choice of permutation function π, context data points $\{(x_i, y_i)\}_{i=1}^N$, and target data points $x_{N+1:N+M}$,

$$p_\theta(y_{N+1:N+M}|x_{N+1:N+M}, x_{1:N}, y_{1:N}) = p_\theta(y_{\pi(N+1):\pi(N+M)}|x_{\pi(N+1):\pi(N+M)}, x_{1:N}, y_{1:N})$$

Proof Outline: The vanilla variant of CMANPs makes predictions similar to that of LBANPs (Feng et al., 2023) by retrieving information from a set of latent vectors via cross-attention. After retrieving information, the final output is computed using an MLP (Predictor). The architecture design of LBANPs ensure that the result is equivalent to making the predictions independently. Since CMANPs use the same querying mechanism as LBANPs, therefore CMANPs preserve target equivariance the same way LBANPs do.

However, for the Autoregressive Not-Diagonal variant (CMANP-AND), the target equivariance is not held as it depends on the order in which the data points are processed. This property is in common with that of several other NP methods by Nguyen & Grover (2022) and Bruinsma et al. (2023).

A.3. Complexity Analysis for CMANP-AND

For a batch of M data points and a prediction block size of b_Q (hyperparameter constant), there are $\lceil \frac{M}{b_Q} \rceil$ batches of data points whose predictions are made autoregressively. Each batch incurs a constant complexity of $\mathcal{O}(b_Q)^2$ due to predicting a full covariance matrix. As such for a batch of M target data points, CMANP-AND requires a sub-quadratic total computation of $\mathcal{O}(\lceil \frac{M}{b_Q} \rceil b_Q^2) = \mathcal{O}(M b_Q)$ with a sequential computation length of $\mathcal{O}(\frac{M}{b_Q})$. Crucially, CMANP-AND only requires constant memory in $|\mathcal{D}_C|$ and linear memory in M, making it significantly more efficient than prior works which required at least quadratic memory.

A.4. Cross Attention Efficient Updates Property is inapplicable to popular attention mechanisms

Cross Attention's efficient updates property allows $\mathrm{CrossAttention}(L, \mathcal{D}_C \cup \mathcal{D}_U)$ to be efficiently computed given $\mathrm{CrossAttention}(L, \mathcal{D}_C)$. This property is conditional on the following: (1) the query tokens remaining the same and (2) the new tokens being stacked on the old context tokens. However, this is not the case for popular attention methods.

For example, the closest to our work is the family of methods based on Perceiver (Jaegle et al., 2021a;b). Perceiver encodes the context dataset via a series of iterative attention blocks. Each iterative attention block consists of a Cross Attention layer to retrieve information from the context dataset for a given set of latents and self-attention layer(s) applied to the latents to compute higher-order information. As a result of stacking these layers, Perceiver fails condition (1) due to the query vectors changing while being stacked.

In brief, Perceiver's iterative attention layers work as follows:

$$L_i = \mathrm{SelfAttention}(\mathrm{CrossAttention}(L_{i-1}, \mathcal{D}_C))$$

where L_i is a fixed-sized set of latents computed iteratively.

When the context dataset is updated with new data $\mathcal{D}_U$, the computation is as follows:

$$L'_i = \text{SelfAttention}(\text{CrossAttention}(L'_{i-1}, \mathcal{D}_C \cup \mathcal{D}_U))$$

To achieve the efficient updates, it requires that $\text{CrossAttention}(L'_{i-1}, \mathcal{D}_C \cup \mathcal{D}_U)$ be efficiently computed from $\text{CrossAttention}(L_{i-1}, \mathcal{D}_C)$. However, $L'_{i-1} \neq L_{i-1}$ for $i > 1$. For example, $L'_1 = \text{SelfAttention}(\text{CrossAttention}(L'_0, \mathcal{D}_C \cup \mathcal{D}_U))$ and $L_1 = \text{SelfAttention}(\text{CrossAttention}(L_0, \mathcal{D}_C \cup \mathcal{D}_U))$.

As such, the efficient updates property of Cross Attention cannot be applied to Perceiver-style architectures.

Another example is that of Transformers. In Transformer Encoders self-attention is repeatedly applied starting from the context tokens, resulting in the outputs of the layers changing drastically when new tokens are added to the context, failing property (2). In the case of the decoder, the argument is the same as Perceiver where the stacked layers cause the queries to change overtime.

B. Appendix: Intuition for Constant Memory Attention Block's construction

In this work, we aim to leverage an efficient attention block for Neural Processes that achieves competitive performance with prior state-of-the-art. As such, the attention block has the following requirements:

1. **Computational Efficiency.** Prior attention-based models require linear or quadratic memory in the number of tokens, limiting their effectiveness in low-resource scenarios. Furthermore, they often require re-computing from scratch when receiving new tokens, i.e., expensive updates. Improving on this, we desire a constant memory version of attention that also allows for constant computation updates.

2. **Permutation invariant in the context.** Neural Processes (NPs) (Garnelo et al., 2018) are permutation invariant in the context.

3. **Stackable.** Modern deep learning leverages the stacking of the same kind of modules (e.g., Transformers) to construct deep models and achieve strong performance.

4. **Computes higher-order information between tokens.** A powerful feature of Transformers that allows it to achieve strong performance.

In Section 3.1, we showed that Cross Attention with a fixed query vector can compute its output in constant memory and can compute updates in constant computation. Notably, this solves the first requirement (Computational Efficiency).

However, since previous attention-based models do not have a fixed query, this efficiency property does not apply to them. As such, we design our own attention block to address these requirements. We begin with a Cross Attention module learned with a fixed query, i.e., $L'_B \leftarrow \text{CrossAttention}(L_B, \mathcal{D}_C)$ where L_B is the fixed query and $\mathcal{D}_C$ is the context data. Since attention is by default permutation-invariant in the context, this module resolves the second requirement (Permutation Invariant), i.e., L'_B is permutation invariant in $\mathcal{D}_C$.

However, this module by itself is not stackable since it comprises of only two inputs: L_B the fixed query (i.e., a learned constant) and $\mathcal{D}_C$ is the context data. Thus to achieve the third requirement (Stackable), we introduce another cross attention block with a new input latent L_I as follows: $L_I^{i+1} \leftarrow \text{CrossAttention}(L_I^i, L'_B)$ where L_I^i is the output of the previous block and L_I^{i+1} is the output of the current block.

Finally, to achieve the fourth requirement (Higher-order information), we wrap the CrossAttention modules with a Self-Attention module. Together, these modules result in the final version of the Constant Memory Attention Block which we proposed in the paper:

$$L'_B \leftarrow \text{SelfAttention}(\text{CrossAttention}(L_B, \mathcal{D}_C))$$

$$L_I^{i+1} \leftarrow \text{SelfAttention}(\text{CrossAttention}(L_I^i, L'_B))$$

	Mooc			Reddit		
	RMSE	NLL	ACC	RMSE	NLL	ACC
THP	0.202 ± 0.017	**0.267 ± 0.164**	**0.336 ± 0.007**	**0.238 ± 0.028**	**0.268 ± 0.098**	**0.610 ± 0.002**
CMHP	**0.168 ± 0.011**	**-0.040 ± 0.620**	0.237 ± 0.024	**0.262 ± 0.037**	**0.528 ± 0.209**	**0.609 ± 0.003**

Table 6. Temporal Point Processes Experiments.

C. Appendix: Additional Experiments and Analyses

C.1. Applying CMABs to Temporal Point Processes (TPPs)

Constant Memory Attention Blocks (1) do not leverage any modality-specific components and (2) are permutation invariant by default like Transformers. As such, CMABs appear to be naturally applicable to a broad range of applications beyond that of Neural Processes. As a proof of concept, we showcase the efficacy of CMABs on next-event prediction (Temporal Point Processes (TPPs)). In brief, Temporal Point Processes are stochastic processes composed of a time series of discrete events. Recent works have proposed to model this via a neural network. Notably, models such as THP (Zuo et al., 2020) encode the history of past events to predict the next event, i.e., modelling the predictive distribution of the next event $p_\theta(\tau_{l+1}|\tau_{\leq l})$ where θ are the parameters of the model, τ represents an event, and l is the number of events that have passed. Typically, an event comprises a discrete temporal (time) stamp and a mark (categorical class).

C.1.1. Constant Memory Hawkes Processes (CMHPs)

Building on CMABs, we introduce the Constant Memory Hawkes Process (CMHPs) (Figure 5), a model which replaces the transformer layers in Transformer Hawkes Process (THP) (Zuo et al., 2020) with Constant Memory Attention Blocks. Unlike THPs which summarise the information for prediction in a single vector, CMHPs summarise it as a set of latent vectors. As such, a flattening operation is added at the end of the model. Following prior work (Bae et al., 2023; Shchur et al., 2020), the predictive distribution for THPs and CMHPs is a mixture of log-normal distribution.

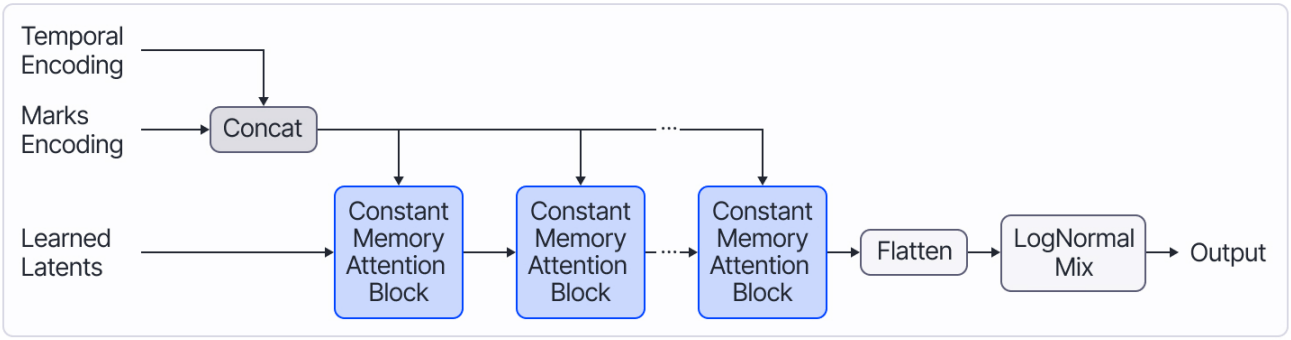


Figure 5. Constant Memory Hawkes Processes

C.1.2. CMHPs: Experiments

In these experiments, we compare CMHPs against THPs on standard TPP datasets: Mooc and Reddit.

Mooc Dataset (Kumar et al., 2019) comprises of $7,047$ sequences. Each sequence contains the action times of an individual user of an online Mooc course with 98 categories for the marks.

Reddit Dataset (Kumar et al., 2019) comprises of $10,000$ sequences. Each sequence contains the action times from the most active users with marks being one of the 984 the subreddit categories of each sequence.

The results (Table 6) suggest that replacing the transformer layer with CMAB (Constant Memory Attention Block) achieves comparable performance. Crucially, CMHP only requires constant memory unlike the quadratic memory required by that of THP. Furthermore, CMHP efficiently updates its model with new data as it arrives over time which is typical in event sequence data, making it significantly more efficient than THP.

Method	$\delta = 0.7$	$\delta = 0.9$	$\delta = 0.95$	$\delta = 0.99$	$\delta = 0.995$
Uniform	100.00 ± 1.18	100.00 ± 3.03	100.00 ± 4.16	100.00 ± 7.52	100.00 ± 8.11
CNP	4.08 ± 0.29	8.14 ± 0.33	8.01 ± 0.40	26.78 ± 0.85	38.25 ± 1.01
CANP	8.08 ± 9.93	11.69 ± 11.96	24.49 ± 13.25	47.33 ± 20.49	49.59 ± 17.87
NP	1.56 ± 0.13	2.96 ± 0.28	4.24 ± 0.22	18.00 ± 0.42	25.53 ± 0.18
ANP	1.62 ± 0.16	4.05 ± 0.31	5.39 ± 0.50	19.57 ± 0.67	27.65 ± 0.95
BNP	62.51 ± 1.07	57.49 ± 2.13	58.22 ± 2.27	58.91 ± 3.77	62.50 ± 4.85
BANP	4.23 ± 16.58	12.42 ± 29.58	31.10 ± 36.10	52.59 ± 18.11	49.55 ± 14.52
TNP-D	**1.18 ± 0.94**	**1.70 ± 0.41**	2.55 ± 0.43	**3.57 ± 1.22**	**4.68 ± 1.09**
LBANP	**1.11 ± 0.36**	**1.75 ± 0.22**	**1.65 ± 0.23**	6.13 ± 0.44	8.76 ± 0.15
CMANP (Ours)	**0.93 ± 0.12**	**1.56 ± 0.10**	**1.87 ± 0.32**	9.04 ± 0.42	13.02 ± 0.03

Table 7. Contextual Multi-Armed Bandit Experiments with varying δ. Models are evaluated according to cumulative regret (lower is better). Each model is run 50 times for each value of δ.

C.2. CMANPs Experiment: Contextual Bandits

In the Contextual Bandit setting introduced by (Riquelme et al., 2018), a unit circle is divided into 5 sections which contain 1 low reward section and 4 high reward sections δ defines the size of the low reward section while the 4 high reward sections have equal sizes. In each round, the agent has to select 1 of 5 arms that each represent one of the regions. For context during the selection, the agent is given a 2-D coordinate X and the actions it selected and rewards it received in previous rounds.

If $||X|| < \delta$, then the agent is within the low reward section. If the agent pulls arm 1, then the agent receives a reward of $r \sim \mathcal{N}(1.2, 0.012)$. Otherwise, if the agent pulls a different arm, then it receives a reward $r \sim \mathcal{N}(1.0, 0.012)$. Consequently, if $||X|| \geq \delta$, then the agent is within one of the four high-reward sections. If the agent is within a high reward region and selects the corresponding arm to the region, then the agent receives a large reward of $N \sim \mathcal{N}(50.0, 0.012)$. Alternatively, pulling arm 1 will reward the agent with a small reward of $r \sim \mathcal{N}(1.2, 0.012)$. Pulling any of the other 3 arms rewards the agent with an even smaller reward of $r \sim \mathcal{N}(1.0, 0.012)$.

During each training iteration, $B = 8$ problems are sampled. Each problem is defined by $\{\delta_i\}_{i=1}^{B}$ which are sampled according to a uniform distribution $\delta \sim \mathcal{U}(0, 1)$. $N = 512$ points are sampled as context data points and $M = 50$ points are sampled for evaluation. Each data point comprises of a tuple (X, r) where X is the coordinate and r is the reward values for the 5 arms. The objective of the model during training is to predict the reward values for the 5 arms given the coordinates (context data points).

During the evaluation, the model is run for 2000 steps. At each step, the agent selects the arm which maximizes its UCB (Upper-Confidence Bound). After which, the agent receives the reward value corresponding to the arm. The performance of the agent is measured by cumulative regret. For comparison, we evaluate the modes with varying δ values and report the mean and standard deviation for 50 seeds.

Results. In Table 7, we compare CMANPs with other NP baselines. We see that CMANP achieves competitive performance with state-of-the-art for $\delta \in \{0.7, 0.9, 0.95\}$. However, the performance degrades as δ reaches extreme values close to the limit such as 0.99 and 0.995 – settings that are at the extremities of the training distribution.

C.3. Additional Analyses

Memory Complexity: In Table 8, we include a comparison of CMANPs with all NP baselines, showing that CMANPs are amongst the best in terms of memory efficiency when compared to prior NP methods. Notably, the methods with a similar memory complexity to CMANPs perform significantly worse in terms of performance across the various experiments (Tables 3 and 4). As such, CMANPs provide the best trade-off regarding memory and performance.

Effect of CMANP's bottleneck on performance: As with any method that uses a bottleneck (e.g., perceiver, set transformer, LBANPs, etc...), CMAB's bottleneck usage results in some amount of information loss. Whether or not this affects the performance is dependent on (1) the amount of information loss (e.g., number of context tokens related to the bottleneck size) and (2) the intrinsic dimensionality of the task (i.e., task complexity).

For example, in a task with a low intrinsic dimensionality, only a small amount of information from the context tokens

	Conditioning	Querying		Updating	
In Terms of	$\lvert\mathcal{D}_C\rvert$	$\lvert\mathcal{D}_C\rvert$	M	$\lvert\mathcal{D}_C\rvert$	$\lvert\mathcal{D}_U\rvert$
CNP	✓	✓	✓	✓	✓
CANP	✗	✗	✓	✗	✗
NP	✓	✓	✓	✓	✓
ANP	✗	✗	✓	✗	✗
BNP	✓	✓	✓	✓	✓
BANP	✗	✗	✓	✗	✗
TNP-D	N/A	✗	✗	N/A	N/A
LBANP	✓	✓	✓	✓	✓
CMANP (Ours)	✓	✓	✓	✓	✓
TNP-ND	N/A	✗	✗	N/A	N/A
LBANP-ND	✓	✓	✗	✓	✓
CMANP-AND (Ours)	✓	✓	✓	✓	✓

Table 8. Comparison of Memory Complexities of Neural Processes with respect to the number of context data points $|\mathcal{D}_C|$, number of target data points in a batch M, and the number of new data points in an update $|\mathcal{D}_U|$. (Green) Checkmarks represent requiring constant memory, (Orange) half checkmarks represent requiring linear memory, and (Red) Xs represent requiring quadratic or more memory.

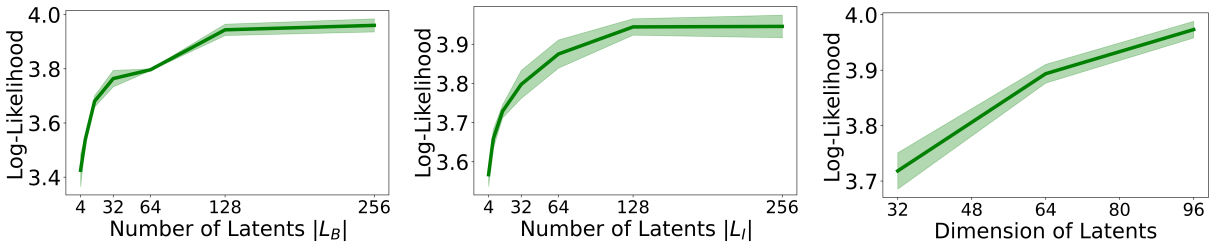


Figure 6. CMANP's bottleneck size on performance. (Left) $|L_B|$'s effect on performance. (Middle) $|L_I|$'s effect on performance. (Right) Latent dimensions' effect on performance.

is needed to solve it. As such, a smaller bottleneck (i.e., low values for L_B and L_I) suffices. However, in a task with a high intrinsic dimensionality, more information is needed from the context tokens. As such, a larger bottleneck (i.e., higher values for L_B and L_I) would be needed to hold the information.

In the main paper (Figure 4 (Right)), we included an analysis of the bottleneck size's effect on task performance where $L_I = L_B$. To analyze this further, we performed an additional three analyses, measuring the performance of the model with respect to individual varying values of L_I, L_B, and latent dimension. In the plots (Figure 6), we see similar results where the performance increases as the bottleneck size increases, saturating slowly. Since the performance generally increases, the bottleneck size should be selected according to the available computational resources. Regardless of the bottleneck sizes that we tried, the model achieves strong performance outperforming all non-attention based models (NP, CNP, and BNP) and several attention-based models (ANP, CANP, and BANP), making it highly applicable to low-resource scenarios.

To analyze the effect of a large number of contexts relative to the number of latents, we additionally run an experiment in the higher resolution setting CelebA (128x128), varying the number of contexts while fixing the number of latents to 128. In the plot (Figure 7), we see that the performance increases as the number of contexts increases, saturating eventually. As more pixels are being added in this experiment, the model naturally receives more relevant information for image completion, resulting in the performance increase. However, since there is ultimately a bottleneck in the model, the performance ultimately saturates given a very large number of contexts.

Empirical Time Comparison with Baselines: In Figure 8, we compare the runtime of CMANPs with various state-of-the-art NP baselines.

In Figure 8a, we compare the runtime of NPs' update phase. Specifically, we find that CMANPs' efficient (fast) update process based on the efficient updates property of Cross Attention only requires constant computation regardless of the

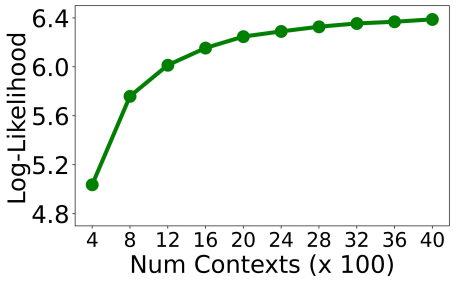
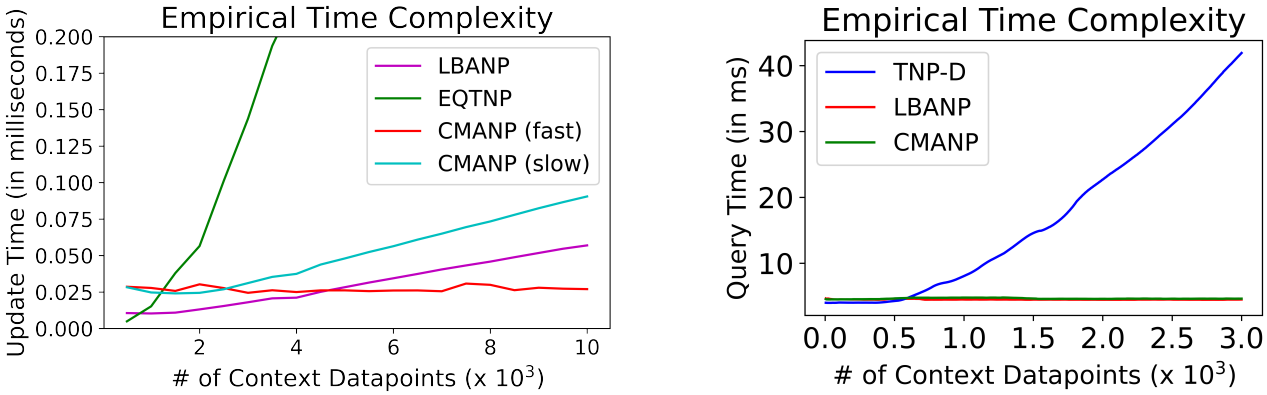


Figure 7. Number of context datapoints on performance.



(a) Empirical runtime of the update phase.

(b) Empirical runtime of the query phase.

Figure 8. Analyses Graphs comparing the runtime of CMANPs with various baselines. (a) Comparison of the update procedure of CMAB-based NP (CMANPs) with Perceiver's iterative attention-based NP model (LBANPs) and a transformer-based NP model (EQTNP). CMANP (fast) refers to the CMAB's efficient update mechanism. CMANP (slow) refers to the traditional update mechanism. (b) Comparison of the query/inference process of CMANPs with LBANPs (Perceiver's iterative attention-based model), TNPs (Transformer-based model), and EQTNPs (Transformer-based model with an efficient query mechanism).

number of context data points. In contrast, the traditional (slow) update process scales linearly with respect to the number of context data points. Furthermore, the Transformer-based (EQTNP) model requires quadratic time complexity and the Perceiver-based (LBANP) model requires linear time complexity. As such, CMANPs scale significantly better than prior state-of-the-art methods.

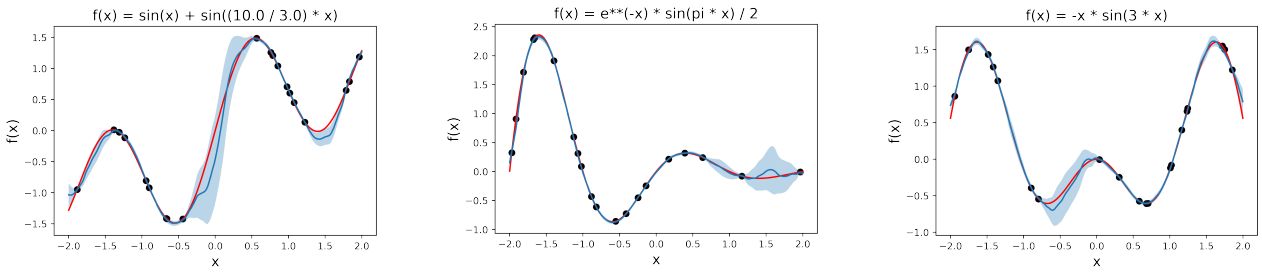
In Figure 8b, we compare the querying (inference) runtime of CMANP with LBANPs (Perceiver's iterative attention-based model), TNPs (Transformer-based model). Notably, TNPs scale quadratically in runtime, making it prohibitively expensive for a large number of context data points. EQTNP (Efficient Queries Transformer Neural Processes) scales linearly. In contrast, CMANPs and LBANPs are the most efficient when performing queries since they are constant complexity (see Table 8) regardless of the number of context data points.

Visualizations: In Figure 9, we show visualizations for the Meta-Regression task. In Figure 10, we show out-of-distribution visualizations for the Meta-Regression task. In Figure 11, we show visualizations for the Image Completion task.

D. Appendix: Discussion

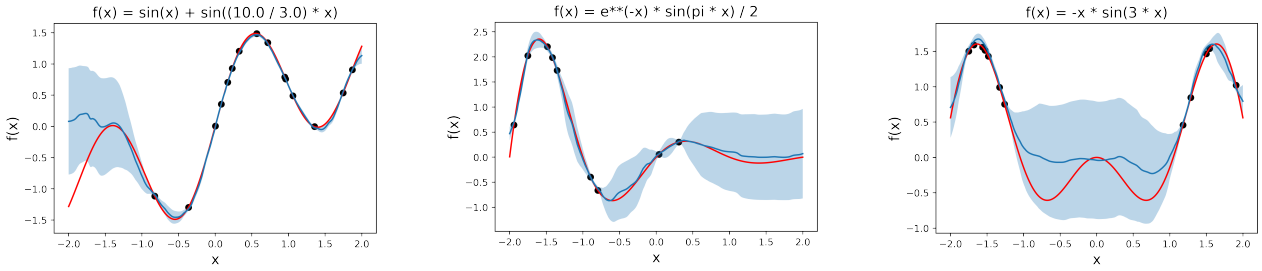
D.1. Likelihood computation of Autoregressive Not-Diagonal extension compared with that of Not-Diagonal extension:

In the Autoregressive Not-Diagonal extension the predictions are made autoregressively, allowing for the modelling of more flexible distributions than prior Not-Diagonal variants. As such, the autoregressive not-diagonal variant's likelihood is typically higher than that of the non-autoregressive baselines which only model an unimodal gaussian distribution.



(a) $y = sin(x) + sin(10/3) * x$ (b) $y = e^{-x} * sin(\pi * x)/2$ (c) $y = -x * sin(3 * x)$

Figure 9. CMANPs Meta-Regression Visualizations.



(a) $y = sin(x) + sin(10/3) * x$ (b) $y = e^{-x} * sin(\pi * x)/2$ (c) $y = -x * sin(3 * x)$

Figure 10. CMANPs Meta-Regression Out-of-Distribution Visualizations. The model is evaluated between $[-2.0, 2.0]$. However, context data points are sampled from only (a) $[-1.0, 2.0]$, (b) $[-2.0, 1.0]$, and (c) $[-2.0, -1.0] \cup [1.0, 2.0]$.

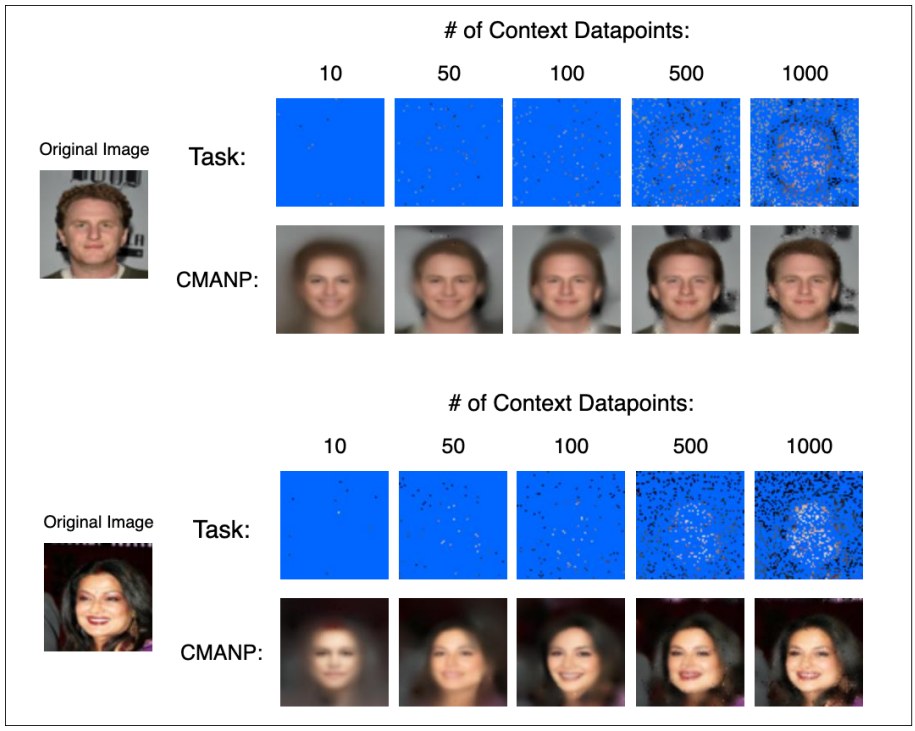


Figure 11. CMANPs Image Completion Visualizations

Consider the following didactic example where $B_Q = 1$ (the block prediction size). Since -AND feeds earlier samples back into the model for making predictions, the likelihood of the target data points: $\{(x_i, y_i)\}_{i=1}^M$ for our -AND model is computed as follows:

$$\begin{aligned}\log p_{AND}(y_{1:M}|x_{1:M}, \mathcal{D}_C) &= \log \prod_{i=1}^{M} p(y_i|x_{1:i-1}, y_{1:i-1}, x_i, \mathcal{D}_C) \\ &= \sum_{i=1}^{M} \log p(y_i|x_{1:i-1}, y_{1:i-1}, x_i, \mathcal{D}_C)\end{aligned}$$

In contrast, consider the likelihood of -ND: $\log p_{ND}(y_{1:M}|x_{1:M}, \mathcal{D}_C)$. By Boole's Inequality (or Union Bound), we have that

$$\log p_{ND}(y_{1:M}|x_{1:M}, \mathcal{D}_C) \leq \sum_{i=1}^{M} \log p(y_i|x_{1:M}, \mathcal{D}_C) = \sum_{i=1}^{M} \log p(y_i|x_i, \mathcal{D}_C)$$

$(x_{1:i-1}, y_{1:i-1})$ provides relevant information for predicting the value of the function at x_i, e.g., nearby pixel values in image completion. As a result, it is likely the case that:

$$p(y_i|x_i, \mathcal{D}_C) \leq p(y_i|x_{1:i-1}, y_{1:i-1}, x_i, \mathcal{D}_C)$$

Summing from $i = 1 \dots M$, this means:

$$\log p_{ND}(y_{1:M}|x_{1:M}, \mathcal{D}_C) \leq \log p_{AND}(y_{1:M}|x_{1:M}, \mathcal{D}_C)$$

As such, the Autoregressive Not-Diagonal variant's likelihood is typically higher than that of the Not-Diagonal baselines (i.e., non-autoregressive variants).

E. Appendix: Implementation, Hyperparameter Details, and Compute

E.1. Implementation and Hyperparameter Details

We use the implementation of the baselines from the official repository of TNPs (https://github.com/tung-nd/TNP-pytorch) and LBANPs (https://github.com/BorealisAI/latent-bottlenecked-anp). The datasets are standard for Neural Processes and are available in the same link. We follow closely the hyperparameters of TNPs and LBANPs. In CMANP, the number of blocks for the conditioning phase is equivalent to the number of blocks in the conditioning phase of LBANP. Similarly, the number of cross-attention blocks for the querying phase is equivalent to that of LBANP. We used an ADAM optimizer with a standard learning rate of $5e-4$. We performed a grid search over the weight decay term $\{0.0, 0.00001, 0.0001, 0.001\}$. Consistent with prior work (Feng et al., 2023) who set their number of latents $L = 128$, we also set the number of latents to the same fixed value $L_I = L_B = 128$ without tuning. Due to CMANPs and CMABs architecture, they allow for varying embedding sizes for the learned latent values (L_I and L_B). For simplicity, we set the embedding sizes to 64 consistent with prior works (Nguyen & Grover, 2022; Feng et al., 2023). The block size for CMANP-AND is set as $b_Q = 5$. During training, CelebA (128x128), (64x64), and (32x32) used a mini-batch size of 25, 50, and 100 respectively. All experiments are run with 5 seeds. For the Autoregressive Not-Diagonal experiments, we follow TNP-ND and LBANP-ND (Nguyen & Grover, 2022; Feng et al., 2023) and use cholesky decomposition for our LBANP-AND experiments. Focusing on the efficiency aspect, we follow LBANPs in the experiments and consider the conditional variant of NPs, optimizing the log-likelihood directly.

E.2. Compute

All experiments were run on a Nvidia GTX 1080 Ti (12 GB) or Nvidia Tesla P100 (16 GB) GPU. Meta-regression experiments took 4 hours to train. EMNIST took 2 hours to train. CelebA (32x32) took 16 hours to train. CelebA (64x64) took 2 days to train. CelebA (128x128) took 3 days to train.