Training and Inference on Any-Order Autoregressive Models the Right Way

Anonymous Author(s)
Affiliation
Address
email

Abstract

Conditional inference on arbitrary subsets of variables is a core problem in probabilistic inference with important applications such as masked language modeling and image inpainting. In recent years, the family of Any-Order Autoregressive Models (AO-ARMs) – which includes popular models such as XLNet – has shown breakthrough performance in arbitrary conditional tasks across a sweeping range of domains. But, in spite of their success, in this paper we identify significant improvements to be made to previous formulations of AO-ARMs. First, we show that AO-ARMs suffer from redundancy in their probabilistic model, i.e., they define the same distribution in multiple different ways. We alleviate this redundancy by training on a smaller set of univariate conditionals that still maintains support for efficient arbitrary conditional inference. Second, we upweight the training loss for univariate conditionals that are evaluated more frequently during inference. Our method leads to improved performance with no compromises on tractability, giving state-of-the-art likelihoods in arbitrary conditional modeling on text (Text8), image (CIFAR10, ImageNet32), and continuous tabular data domains.

1 Introduction

Generative modeling has seen tremendous progress in building highly expressive models [2, 30], but relatively little effort has been put into supporting efficient probabilistic inference. Most existing models with deep neural architectures do not admit efficient inference on conditional queries of the form $p(\boldsymbol{x}_u|\boldsymbol{x}_v)$, where u and v are disjoint subsets of the variables of the joint distribution. Such queries, however, have many important applications such as masked language modeling [41], image inpainting [42], and more. For example, multi-modal models learn a joint distribution over all data modalities, and at test time may only be presented with some subset (unknown in advance) of modalities [40, 19]. In robot shared-autonomy, an operator at test time may choose to provide a subset of the action inputs, leaving the model to fill in the remaining action dimensions [7, 22].

Evidently, expressive generative models that can support efficient conditional inference can bring progress to many applications. Towards this end, the family of Any-Order Autoregressive Models (AO-ARMs) [36, 41, 34, 13] has shown surprising success. AO-ARMs are built on the following insights. An autoregressive model defines an ensemble of univariate conditionals $p(x_t|\boldsymbol{x}_{<t})$ that leads to efficient inference on certain conditional queries: ones whose variables form a prefix of the ordering (Fig. 1a). However, in standard autoregressive models, all other conditionals are defined only implicitly through Bayes' rule, and hence do not admit efficient inference routines. To fix this, consider training another autoregressive model on the reverse ordering, which now defines a larger

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

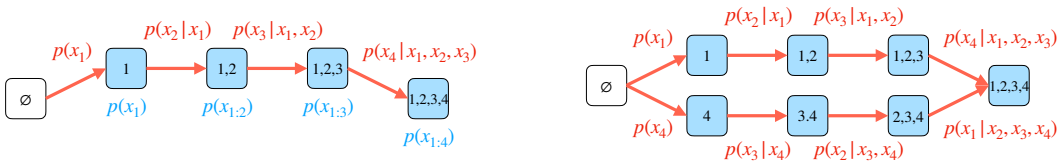


Figure 1: (a) An autoregressive model defines N univariate conditionals $p(x_t|\boldsymbol{x}_{<t})$, allowing for efficient conditional inference when variable subsets form a prefix of the ordering, e.g., $p(\boldsymbol{x}_{1:3})$. (b) Learning an additional autoregressive model on the reverse ordering x_4, x_3, x_2, x_1 enables efficient inference on more queries: when variables form suffixes, e.g., $p(\boldsymbol{x}_{2:4})$. However, this leads to redundancy in the probabilistic model, as the joint distribution $p(\boldsymbol{x}_{1:4})$ is now defined in two different ways. This redundancy is exacerbated in AO-ARMs as a result of learning on all possible orders.

set of univariate conditionals that can answer queries of either the prefix or the suffix of the ordering (Fig. 1b). AO-ARMs take this insight to the extreme by training on all possible orderings, enabling efficient inference on all possible conditionals. Remarkably, recent efforts in scaling up AO-ARMs have worked extremely well, with breakthrough performance in masked language modeling [41], image inpainting [13], and more [34].

However, in spite of their already strong empirical results, we identify significant improvements that can be made to AO-ARMs. We begin by showing that previous formulations of AO-ARMs suffer from redundancy in their probabilistic model. Consider again the example of training two autoregressive models on the lexicographical and reverse ordering (Fig. 1b). This already unnecessarily defines the same joint distribution in two different ways, as there are two paths to the node $p(\boldsymbol{x}_{1:4})$ corresponding to two distinct factorizations. By scaling up to all possible orderings, AO-ARMs inadvertently model the same distribution with a huge amount of redundancy. Such redundancy makes training inefficient and, much more importantly, leads to worse asymptotic performance. Due to finite capacity of the model, attempting to learn too many univariate conditionals can lead to a poor fit on individual ones.

In this paper, we present **MAC** – Mask-tuned Arbitrary Conditional Models – which proposes two key insights for reducing redundancy and improving model performance. The first insight is that arbitrary conditional inference can still be computed efficiently *without* relying on all possible univariate conditionals. For example in Fig. 1b, omitting the edge $p(x_1|x_2, x_3, x_4)$ still maintains efficient inference for all the nodes in the graph, since there is another path for computing $p(\boldsymbol{x}_{1:4})$. Based on this observation, we reduce redundancy of AO-ARMs by training on a smaller set of univariate conditionals that still supports tractable arbitrary conditional inference. The second insight is that some univariate conditionals are evaluated much more often than others during inference. Therefore, we upweight the training loss for the more frequently occurring univariate conditionals, thereby aligning the training and inference objectives more closely.

Combining these insights, MAC leads to a strictly-improved formulation of AO-ARMs that, amazingly, gives better arbitrary conditional and joint likelihoods with *no compromises* on tractability. We achieve state-of-the-art likelihoods for arbitrary conditional modeling across multiple domains such as text (Text8), image (CIFAR10, ImageNet32), and continuous tabular data. Finally, we conclude by demonstrating a novel application of AO-ARMs to robot shared-autonomy.

2 Background

Problem definition Marginal and conditional inference are important queries for handling partial evidence on arbitrary subsets of variables, which is relevant for tasks such as masked language modeling or image inpainting. We let a **mask** e be a subset of variables, and its **cardinality** $|e|$ be the number of elements in the subset. A masked input $\boldsymbol{x}_e$ is a partial instantiation on the variable subset e using the values taken on by input $\boldsymbol{x}$. Marginal inference refers to queries of the form $p(\boldsymbol{x}_e) = \int p(\boldsymbol{x}_e \boldsymbol{x}_q) d\boldsymbol{x}_q$, where $q = X \setminus e$ and $\boldsymbol{x}_e \boldsymbol{x}_q$ denotes the union of the partial instantiations.

Conditional inference queries $p(\boldsymbol{x}_u|\boldsymbol{x}_e) = p(\boldsymbol{x}_u\boldsymbol{x}_e)/p(\boldsymbol{x}_e)$ are similar and can be computed as a ratio of two marginals, so we will refer to marginals and conditionals interchangeably. To evaluate model performance on marginal inference queries over a test dataset X_{test}, we assume a test mask distribution M over the 2^N possible masks. We sample masks e from M independently from data, and use the negative marginal log-likelihood of the masked input $\boldsymbol{x}_e$ as the loss.

$$\mathcal{L}_M = -\sum_{\boldsymbol{x}\in X_{\text{test}}} \mathbb{E}_{e\sim M} \log p(\boldsymbol{x}_e) \tag{1}$$

Computing arbitrary marginals and conditionals has been a long-standing challenge in probabilistic inference, with a rich history of techniques [27, 25, 14, 5, 39] ranging from belief propagation, variational inference, to MCMC. Such queries are inherently difficult in high-dimensions: given access to joint likelihoods $p(\boldsymbol{x})$, naïve computation of marginal likelihoods $p(\boldsymbol{x}_e)$ requires integration over the missing $N - |e|$ dimensions. As a result, most traditional approaches model the joint distribution using non-neural architectures that admit efficient inference but are less expressive [17].

Autoregressive Models Autoregressive models are an influential family of generative models that represents complex high-dimensional distributions by modeling a single dimension at a time. They parameterize a joint distribution $p(\boldsymbol{x})$ over N dimensions by factorizing it into univariate conditionals $\prod_{t=1}^{N} p(x_{\sigma(t)}|\boldsymbol{x}_{\sigma(<t)};\sigma)$ via chain rule (Fig. 1a), using an ordering σ of the N variables. We write $\sigma(t)$ and $\sigma(<t)$ to denote the masks corresponding to the t-th element and the first $t-1$ elements of the ordering, respectively.

The univariate conditionals can be learned via a weight-tied neural network, and composing them together leads to highly expressive architectures in practice, such as PixelCNN [37, 31] or Transformers [38]. Autoregressive models support efficient inference on the joint distribution, and on a specific type of marginal query: ones where the mask forms a prefix of the variable order, i.e., $\sigma(\leq |e|) = e$ (Fig. 1a). We will call such a mask e and ordering σ *compatible*.

However, autoregressive models cannot support efficient arbitrary marginal inference in general. Next, we present AO-ARMs, which extend autoregressive models in a way that does support arbitrary marginal inference.

2.1 Any-Order Autoregressive Models (AO-ARMs)

AO-ARMs [36, 34, 13] learn a single model that can generate the joint distribution autoregressively using any ordering of the N variables, by modeling $\prod_{t=1}^{N} p(x_{\sigma(t)}|\boldsymbol{x}_{\sigma(<t)};\sigma)$ for all orderings σ. Even though there are $N!$ different orderings, their chain-rule factorization is built from univariate conditionals, of which there are "only" $N2^{N-1}$. Therefore, an AO-ARM is a model that defines the $N2^{N-1}$ distinct univariate conditionals $p(x_j|\boldsymbol{x}_{e\setminus j})$, where $j \in e$.

To answer a marginal inference query $p(\boldsymbol{x}_e)$ with an AO-ARM, we choose an order σ that is compatible with e, so that $\sigma(\leq |e|) = e$. Then, we simply evaluate each of the univariate conditionals $p(\boldsymbol{x}_e) = \prod_{t=1}^{|e|} p(x_{\sigma(t)}|\boldsymbol{x}_{\sigma(<t)};\sigma)$ in the autoregressive factorization of $\boldsymbol{x}_e$.

Training AO-ARMs Architecturally, an AO-ARM models all univariate conditionals via a weight-tied neural network, by using a special (so-called "absorbing-state" [1, 13]) token for variables that are not present in the evidence set. To enable parallel optimization, the AO-ARM architecture is designed so that given evidence $\boldsymbol{x}_e$, the model can predict the univariate conditionals $p(x_i|\boldsymbol{x}_e)$ for all $i \in X \setminus e$ at once. Importantly, this parallelization works on non-causal architectures, opening the doors to architectures similar in flexibility to diffusion models [32, 33, 12].

In previous works [36, 34, 13], AO-ARMs are trained to maximize the joint likelihood of a datapoint $\boldsymbol{x}$ under the expectation over the uniform distribution $\mathcal{U}_\sigma$ of orders.

$$\log p(\boldsymbol{x}) = \log \mathbb{E}_{\sigma\sim\mathcal{U}_\sigma} \sum_{t=1}^{N} p(x_{\sigma(t)}|\boldsymbol{x}_{\sigma(<t)};\sigma) \geq \mathbb{E}_{\sigma\sim\mathcal{U}_\sigma} \sum_{t=1}^{N} \log p(x_{\sigma(t)}|\boldsymbol{x}_{\sigma(<t)};\sigma) \tag{2}$$

This objective can be simplified by treating t as a random variable with a uniform distribution $\mathcal{U}_t$ over cardinalities 1 to N. We let $M_{\text{card-edge}}$ be the distribution over the tuple $(\sigma(t), \sigma(< t))$ where $\sigma \sim \mathcal{U}_\sigma$ and $t \sim \mathcal{U}_t$, giving the following loss function for an AO-ARM parameterized by θ.

$$\mathcal{L}(\theta) = -\mathbb{E}_{\sigma \sim \mathcal{U}_\sigma} \sum_{t=1}^{N} \log p_\theta(x_{\sigma(t)} | \boldsymbol{x}_{\sigma(<t)}; \sigma) = -N \cdot \mathbb{E}_{i, e \sim M_{\text{card-edge}}} \log p_\theta(x_i | \boldsymbol{x}_e) \tag{3}$$

In practice, due to parallel optimization, we write the objective in the following equivalent form.

$$\mathcal{L}(\theta) = -N \cdot \mathbb{E}_{-, e \sim M_{\text{card-edge}}} \frac{1}{N - |e|} \sum_{i \in X \setminus e} \log p_\theta(x_i | \boldsymbol{x}_e) \tag{4}$$

Compared to other arbitrary conditional models, AO-ARMs consistently give state-of-the-art performance on benchmarks across a range of domains [36, 34, 13]. Besides good empirical performance, they also serve as a conceptual unification between autoregressive models and diffusion models [13, 1], showing promise for both practical and theoretical advancements.

3 Improving Any-Order Autoregressive Models with MAC

Despite the success of AO-ARMs, in this section we identify two key deficiencies in previous formulations of AO-ARMs. We propose solutions to them by re-interpreting AO-ARMs from the perspective of recursive decomposition over marginals. Our resulting method MAC leads to consistent improvements and state-of-the-art likelihoods across a range of domains in Sec. 4.

(A) The first deficiency is that of redundancy. While being order-agnostic is exactly what enables arbitrary conditionals, learning the same distribution with multiple orderings makes training inefficient. More importantly, since the model has limited capacity (especially with the weight-tied architecture), learning orderings redundantly hurts asymptotic performance.

(B) The second deficiency stems from the AO-ARM training objective in Eq. 2. This objective focuses only on the joint likelihood, leading to a mismatch with the marginal likelihood evaluation objective in Eq. 1, even though computing marginals is the key feature of AO-ARMs.

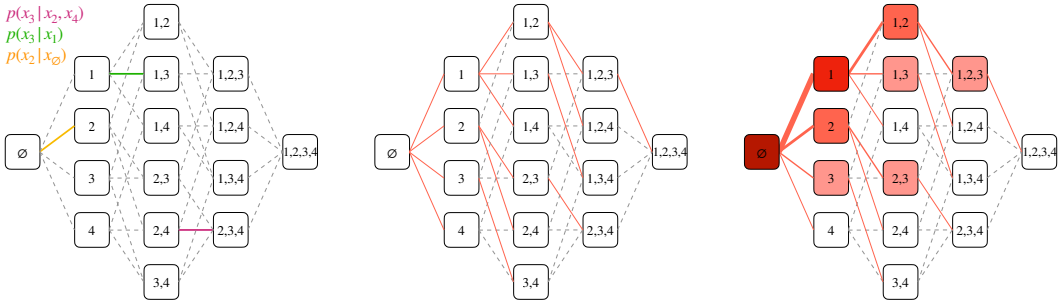
When viewing AO-ARMs as autoregressive models that generate using all orders, it is not obvious how to fix the above two deficiencies. For example, if we try to omit certain orders to address **(A)**, we may lose the ability to efficiently answer marginals, e.g., queries that are prefixes of the orders we removed. Similarly for **(B)**, it is unclear how to choose a distribution of orderings to best account for arbitrary conditional inference. To solve these issues, we re-interpret AO-ARMs from the perspective of computing arbitrary conditional probabilities via recursive decomposition over masks.

3.1 Re-interpreting AO-ARMs as recursive decomposition on a binary lattice

A mask e is associated with the set of marginal inference queries of the form $p(\boldsymbol{x}_e)$. We will refer to this set of queries as the corresponding **task** for mask e. We represent these masks/tasks as nodes in Fig. 1. Supporting efficient computation of each task, therefore, implies efficient arbitrary marginal inference. However, for masks with high cardinality, the corresponding task involves learning a high-dimensional distribution.

Fortunately, these masks do not actually form standalone tasks, but are subproblems of one another. In particular, given non-empty mask e and a single variable $j \in e$, we can write $p(\boldsymbol{x}_e)$ as $p(\boldsymbol{x}_{e \setminus j}) p(x_j | \boldsymbol{x}_{e \setminus j})$. Then, we delegate the computation of $p(\boldsymbol{x}_{e \setminus j})$ to the task associated with mask $e \setminus j$, and are only left with estimating the univariate conditional $p(x_j | \boldsymbol{x}_{e \setminus j})$.

From this perspective, we can view AO-ARMs as solving an ensemble of intertwined tasks, one for each of the 2^N possible masks. We visualize this as a binary lattice in Fig. 2a over $N = 4$ variables, where we have $2^4 = 16$ nodes representing the possible masks/tasks, with an edge connecting nodes



(a) (b) (c)

Figure 2: (a) For $N = 4$, marginal queries can take on one of $2^4 = 16$ possible masks, represented as square nodes. These masks are related to one another through univariate conditionals (edges), since $p(x_e) = p(\boldsymbol{x}_{e\setminus j})p(x_j|\boldsymbol{x}_{e\setminus j})$. For example, given $p(x_1)$, to compute $p(\boldsymbol{x}_{1,3})$ we only need to learn the univariate conditional $p(x_3|x_1)$, shown as the green edge. (b) We don't need to learn all univariate conditionals (edges) to compute arbitrary marginals (nodes). It suffices to learn one left-going edge for every node besides the $\emptyset$ node. (c) Given a marginal query at a node, we recursively decompose it by following red edges until we reach the $\emptyset$ node. Hence, some edges / nodes will be traversed more often, so we should train on them more often.

e' and e if $e' = e \setminus j$ for some singleton variable $j \in e$. To answer a marginal query $p(\boldsymbol{x}_e)$, we choose an element $j \in e$ and compute $p(\boldsymbol{x}_e) = p(\boldsymbol{x}_{e\setminus j})p(x_j|\boldsymbol{x}_{e\setminus j})$, which corresponds to moving along the edge from node e to node $e \setminus j$ in the lattice. We proceed recursively on $p(\boldsymbol{x}_{e\setminus j})$ until we reach $p(\boldsymbol{x}_\emptyset) = 1$.

Under this recursive decomposition framework, two key concepts are the decomposition protocol, which dictate how edges are chosen, and the mask/edge distributions, which specify the probability that different marginals and univariate conditionals are evaluated.

Definition 1 *A **decomposition protocol** w defines, for each non-empty mask e, a probability distribution w_e over the elements in e. In other words, $\sum_{j\in e} w_e(j) = 1$ and $w_e(j) \geq 0$ for $e \neq \emptyset$.*

A decomposition protocol w specifies a process for answering marginal inference queries. Given a marginal query at node e, we sample an element $j \sim w_e$ and compute $p(\boldsymbol{x}_e) = p(\boldsymbol{x}_{e\setminus j})p(x_j|\boldsymbol{x}_{e\setminus j})$. We proceed recursively on $p(\boldsymbol{x}_{e\setminus j})$ until we reach $p(\boldsymbol{x}_\emptyset) = 1$.

Definition 2 *A **mask distribution** is a probability distribution over the 2^N possible variable subsets for marginals. An **edge distribution** is a probability distribution over the $N2^{N-1}$ possible variable subsets for univariate conditionals.*

Mask and edge distributions are relevant for specifying the *test mask distribution* (Eq. 1) of the problem domain, and the *training edge distribution* for the AO-ARM. When viewed under this recursive decomposition interpretation, previous formulations of AO-ARMs use a random decomposition protocol w-RND where $w\text{-RND}_e(j) = 1/|e|$. Moreover, they use a training edge distribution $M_{\text{card-edge}}$ (Eq. 3), which unfortunately neglects to take into account the test mask distribution M.

3.2 MAC: Mask-tuned Arbitrary Conditional Model

We can now describe our method MAC. MAC improves upon previous AO-ARMs by choosing a custom decomposition protocol and deriving a training edge distribution that aligns with the test mask distribution M.

(A) Reducing edge redundancy To answer arbitrary marginals, we don't need to learn to generate using all possible orderings. Rather, it suffices to decompose each task into any one of its children tasks. This constraint corresponds to learning a single (red) incoming edge for each node besides the $\emptyset$ node. We show an example in Fig. 2b, where we connect each node to its leftward parent obtained

by removing the greatest element of the node's mask. This is precisely the decomposition protocol that we use for MAC: w-MAC$_e(j) = 1$ if j is the greatest element in e.

(B) Matching edge distribution When presented with a marginal query e, the decomposition protocol w traverses a path from node e to node $\emptyset$, evaluating the univariate conditionals along the way. Simulating w on masks $e \sim M$ leads to an *induced edge distribution* $D_{M,w}$ that represents how frequently different univariate conditionals are evaluated during the inference process, which we depict in Fig. 2c using red edges with different thicknesses. We can see, for example, that under the protocol w-MAC, univariate conditionals such as $p(x_1|\boldsymbol{x}_\emptyset)$ are evaluated highly frequently because it appears as a subproblem to many marginal queries. Therefore, we should tune our training distribution on univariate conditionals to match this induced edge distribution $D_{M,w}$ given by the downstream test mask distribution M and the decomposition protocol w-MAC.

Abstractly, let $\pi \sim w(e)$ denote the probability that w chooses a path of edges $\pi = \{(i_t, e_t)\}_{t=1}^{|e|}$ when decomposing e. In particular, $e_1 = \emptyset$, and i_t is a variable in $e \setminus e_t$, and $e_{t+1} = i_t \cup e_t$. Then we can write the marginal inference objective as follows, where C is the constant $\mathbb{E}_{i,e\sim D_{M,w}}|e|$.

$$\mathbb{E}_{e\sim M}[\log p(\boldsymbol{x}_e)] = \mathbb{E}_{e\sim M}\mathbb{E}_{\pi\sim w(e)} \sum_{t=1}^{|e|} \log p(x_{i_t}|\boldsymbol{x}_{e_t}) \tag{5}$$

$$= C \cdot \mathbb{E}_{i,e\sim D_{M,w}} \log p(x_i|\boldsymbol{x}_e) \tag{6}$$

Eq. 5 leads to the following algorithms for training MAC (Alg. 1). We sample a marginal query $p(\boldsymbol{x}_e)$ by independently sampling a mask $e \sim M$ and an input $\boldsymbol{x} \sim X$. Then, we simulate the path taken on the lattice by the decomposition protocol w by iteratively sampling an element $j \sim w_e$ and updating e to be $e \setminus j$. Throughout this process, we optimize our model on the edges (univariate conditionals) traversed on this path. During testing (Alg. 2), we similarly step through the decomposition protocol and add together the univariate conditional likelihoods along the path.

Algorithm 1: Training MAC

Input: Test mask distribution M, decomposition protocol w, training data distribution X, model θ

while *training* **do**
  $e \sim M, \boldsymbol{x} \sim X$
  while $e \neq \emptyset$ **do**
    $j \sim w_e$
    $\theta \leftarrow \theta + \nabla_\theta \log p_\theta(x_j|\boldsymbol{x}_{e\setminus j})$
    $e \leftarrow e \setminus j$

Algorithm 2: Testing MAC

Input: Mask e, decomposition protocol w, test data $\boldsymbol{x}$, model θ

Output: Marginal log-likelihood $\log p_\theta(\boldsymbol{x}_e)$

$r \leftarrow 0$
while $e \neq \emptyset$ **do**
  $j \sim w_e$
  $r \leftarrow r + \log p_\theta(x_j|\boldsymbol{x}_{e\setminus j})$
  $e \leftarrow e \setminus j$

Return: r

In practice, since the decomposition protocol w-MAC deterministically removes the greatest element, there is an efficient way to directly sample from $D_{M,w\text{-MAC}}$ in batch (Eq. 6), without having to simulate w-MAC as in Alg. 1. We describe these details in the Appendix. Finally, to conclude this section, we discuss two important design choices and practical considerations for w and $D_{M,w}$.

Parallel training As noted in Section 2, the architecture of AO-ARMs enables training on univariate conditionals of the form $p(x_i|\boldsymbol{x}_e)$ for $i \in X \setminus e$ in parallel. Therefore, rather than dealing with a training distribution over univariate conditionals $D_{M,w}(i, e)$, we instead work with one over masks $D_{M,w}(e)$. To a close approximation, we will simply let $D_{M,w}(e) \propto \sum_i D_{M,w}(i, e)$, visualized as shaded nodes in Fig. 2c.

Cardinality Reweighting (heuristic) We present one additional technique for tuning the training mask distribution, with ablations and further discussion in later sections. To foster generalization of our weight-tied neural network, we reweigh the probability of a mask based on its cardinality e, by using a final training distribution of $D^{\text{CR}}_{M,w}(e) \propto (1 + |e|)D_{M,w}(e)$, which we sample using SIR (Sampling-Importance-Resampling).

With parallelism and cardinality reweighting, the final objective for MAC is the following:

$$\mathcal{L}_M(\theta) := -\mathbb{E}_{e\sim D^{\text{CR}}_{M,w\text{-MAC}}} \frac{1}{N-|e|} \sum_{i\in X\setminus e} \log p_\theta(x_i|\boldsymbol{x}_e) \tag{7}$$

The decomposition protocol w-MAC recursively removes the greatest element of a mask, where comparison between two elements $j_1 > j_2$ is determined by a global canonical ordering (e.g. lexicographical ordering).

4 Experiments

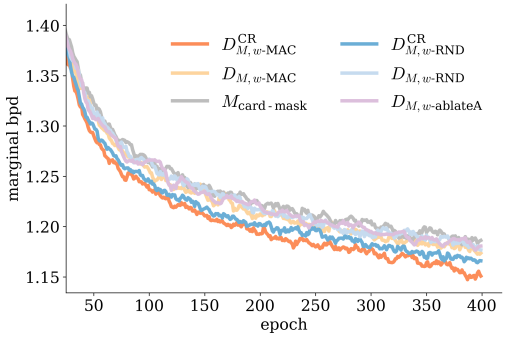
We evaluate MAC on high-dimensional language and image domains, and on a set of continuous tabular benchmarks. We focus on two metrics: joint likelihood and marginal likelihood of the test set. On both metrics, MAC shows state-of-the-art performance among arbitrary conditional models on the majority of benchmarks. We conclude by demonstrating a novel application of arbitrary conditional models on a simulated robot shared-autonomy task, where we learn a conditional policy in a 9-DoF action space through implicit behavioral cloning with MAC.

Note on computation Each run was done on a single NVIDIA A40. For the language and image experiments, we trained each model for approximately two weeks. Although this was not enough to match the total number of epochs trained by the baseline ARDM[1], we were still able to show state-of-the-art performance for arbitrary conditional models on 2 out of the 3 language/image benchmarks, and beat the baselines on all 3 benchmarks when compared under the same number of training epochs.

Table 1: Character-level modeling of the Text8 dataset (without additional context), in bpd.

	joint	marginal
Joint Models		
from literature		
Transformer [38]	1.35	–
Arb. Cond. Models		
from literature		
OA-Transformer [13, 41]	1.64	–
D3PM [1]	1.47	–
ARDM [13] (14000 epochs)	1.43	–
our experiments		
ARDM (3000 epochs)	1.48	1.12
MAC (3000 epochs)	1.40	1.09

Figure 3: Ablation on choices of training mask distribution. The best performing mask distribution $D^{\text{CR}}_{M,w\text{-MAC}}$ uses the w-MAC decomposition protocol with cardinality reweighing.



4.1 Language

We learn a character-level model on chunks of 250 characters using the Text8 dataset [23], which consists of 100M characters. We follow the setup from previous work in modeling the chunked text segments *without* any additional context, and use the same 12-layer Transformer architecture as ARDM and D3PM [13, 1]. We report the joint and marginal likelihoods in terms of bpd (lower is better), where the marginal bpd is computed w.r.t. to the test mask distribution $M_{\text{card-mask}}(e) \propto 1/\binom{N}{|e|}$ as used in [34][2]. In Tab. 1 we see that MAC outperforms all existing arbitrary conditional models in joint likelihood and comes close to a standard Transformer trained only to model in one variable order, even when trained only on a fraction of the total number of epochs by baselines in literature. Similarly, MAC also gets better marginal bpd compared to ARDM.

[1] Previous works were unable to share checkpoints of baseline methods.

[2] $M_{\text{card-mask}}$ effectively samples a cardinality c, and then samples uniformly among masks with cardinality c.

Ablations In Fig. 3 we present ablation studies on different choices of the training mask distribution. The baseline method (ARDM) trains on the $M_{\text{card-mask}}$ mask distribution (grey). Using only insight **(A)** from Sec. 3 with a deterministic decomposition protocol but no edge-weighting gives slight improvements $D_{M,w\text{-ablateA}}$ (light purple). Using only insight **(B)** from Sec. 3 with edge-weighting but random decomposition protocol also leads to slight improvements $D_{M,w\text{-RND}}$ (light blue). Using both insights **(A)** and **(B)** gives the w-MAC protocol which leads to more improvements $D_{M,w\text{-MAC}}$ (light orange). Lastly, the cardinality reweighing heuristic improves both methods even more (dark blue/orange). The best performance is given by $D^{\text{CR}}_{M,w\text{-MAC}}$ in dark orange, which is the setting we use for MAC in all our other experiments.

4.2 Images

We evaluate MAC on CIFAR10 [18] and ImageNet32 [4, 6], both of which consist of images of dimension 3072. Again, we follow all experimental settings from previous work [1, 13, 16], using a U-Net with 32 ResNet Blocks interleaved with attention layers, and use rotation/flip data augmentation. In Tab. 2&3 we see that MAC outperforms ARDM on joint bpd and marginal bpd, evaluated on the $M_{\text{card-mask}}$ mask distribution. Surprisingly, MAC even gives the best joint bpd on ImageNet32 when compared to joint models that do not support arbitrary conditionals.

Table 2: Pixel modeling of the ImageNet32 dataset (with no data augmentation), in bpd.

	joint	marginal
Joint Models		
from literature		
Image Transformer [26]	3.77	–
VDM [16]	3.72	–
Arb. Cond. Models		
our experiments		
ARDM (16 epochs)	3.60	2.10
MAC (16 epochs)	3.58	2.08

Table 3: Pixel modeling of the CIFAR10 dataset (with rotation/flip data augmentation), in bpd.

	joint	marginal
Joint Models		
from literature		
PixelCNN++ [31]	2.88	
Sparse Transformer [3, 15]	2.56	–
VDM [16]	2.49	–
Arb. Cond. Models		
from literature		
D3PM [1]	3.44	–
ARDM [13] (3000 epochs)	2.69	–
our experiments		
ARDM (1200 epochs)	2.86	1.84
MAC (1200 epochs)	2.81	1.81

4.3 Continuous Tabular Data

Next, we consider a tabular domain with variables taking on continuous values [34]. The main challenge with continuous values for AO-ARMs is the parameterization of the univariate conditionals, since 1-D Gaussians or even mixtures of Gaussians have limited expressiveness. To this end, ACE [34] proposes to parameterize the 1-D conditionals as EBMs. We build MAC on top of ACE, keeping all hyperparameters and experimental setup the same, modifying only the training mask distribution. In Tab. 4&5, we see that MAC gives better marginal and conditional likelihoods than ACE on most settings, and outperforms other arbitrary conditional models such as SPNs [29, 24] and ACFlow [21].

Table 4: Marginal log-likelihood on 5 continuous tabular benchmarks (higher is better). The mask cardinality settings kept consistent with the ones reported in [34].

	power			gas			hepmass			miniboone			bsds		
Mask cardinality	3	5	6	3	5	8	3	5	10	3	5	10	3	5	10
SPFlow [24]	-0.63	1.01	-0.12	0.68	1.88	4.81	-4.01	-6.58	-13.38	-2.21	-4.31	-9.85	-2.87	-4.42	-8.15
ACFlow [21]	-0.57	1.34	0.42	0.78	3.01	10.13	-4.03	-6.19	-11.58	-2.76	-5.31	-10.36	5.06	9.26	19.60
ACE [34]	-0.56	1.42	0.58	1.31	4.31	12.20	**-4.00**	-5.91	-10.72	-2.13	-3.80	-7.94	**5.10**	**9.37**	20.31
MAC	**-0.55**	**1.43**	**0.61**	**1.59**	**4.98**	**13.02**	**-4.00**	**-5.90**	**-10.69**	**-2.12**	**-3.76**	**-7.76**	**5.10**	**9.37**	**20.33**

Table 5: Conditional log-likelihood on 5 continuous tabular benchmarks.

	power	gas	hepmass	miniboone	bsds
SPFlow [24]	-1.03	4.30	-12.78	-18.34	-24.15
ACFlow [21]	0.56	8.09	-8.20	-0.97	81.83
ACE [34]	0.63	9.64	-3.86	**0.31**	**86.70**
MAC	**0.65**	**9.77**	**-3.05**	0.07	86.05

Table 6: Shared autonomy on FrankaKitchen with BC operator policy. Full autonomy baseline (IBC) [9]: 2.15 ± 0.06

Conditional Policy	Reward
BC	1.81 ± 0.08
MAC	$\mathbf{2.00} \pm \mathbf{0.05}$

4.4 Shared Autonomy on FrankaKitchen

Finally, we demonstrate an application of arbitrary conditional models on the task of shared autonomy in robotic manipulation. In shared autonomy settings, an operator aims to control a policy in a complicated action space, but is allowed to delegate partial control to an AI model for easier manipulation. We focus on the simulated FrankaKitchen environment [11, 10], where the goal is to control a 9-DoF robotic arm to move objects around a kitchen.

We train a base policy with behavioral cloning (BC) [28] and an arbitrary conditional policy with MAC to solve the `Kitchen-mixed0` task. To simulate shared autonomy, we evaluate a hybrid policy where at each state, the operator policy (BC) outputs 4 out of the 9 dimensions of the action space, and a conditional policy fills in the remaining 5 action dimensions. In Tab. 6 we see that conditioning using MAC improves shared autonomy as compared to conditioning using an independent BC policy, and comes close to the full autonomy performance [9].

5 Discussion

Sharpen mask distribution Intuitively, we should choose a protocol w such that $D_{M,w}$ has low entropy so that our model has a greater chance of revisiting similar masks. To investigate the differences in choices of decomposition protocol, in Table 7 we present Monte-Carlo simulations of the induced mask distributions with respect to the test mask distribution $M_{\text{card-mask}}$. As expected, using w-MAC leads to lower empirical entropy than using w-RND.

Table 7: Monte-Carlo simulation of induced mask distributions for $N = 12$, taking 1e6 samples. For brevity we only display a few selected rows.

Mask	M	$D_{M,w\text{-RND}}$	$D_{M,w\text{-MAC}}$	$D^{\text{CR}}_{M,w\text{-MAC}}$
`011111111110`	1261	222	650	1827
`100000000000`	7037	14803	76381	41441
`100000000101`	407	521	0	0
`100000001000`	1288	1975	536	430
`111111111110`	6956	433	6935	21094
`Entropy`	7.21	5.74	4.44	5.51

Mask generalization (and why we do Cardinality Reweighting) The success of AO-ARMs depends on generalization to unseen masks, since there are exponentially many masks. Training on many masks improves generalization, and can be viewed as a form of data augmentation that allows AO-ARMs to sometimes even outperform standard autoregressive models on joint likelihoods.

In light of this, a decomposition protocol w that skews the mask distribution may have the unfortunate side-effect of hurting generalization. In particular, the mask distribution $D_{M,w\text{-MAC}}$ puts much more weight on low-cardinality masks. This is problematic since training on cardinality-1 masks is not conducive to generalization: there are only N possible cardinality-1 masks, whereas higher cardinalities have exponentially more masks. As such, we proposed $D^{\text{CR}}_{M,w\text{-MAC}}$ (Table 7) with the hypothesis that focusing less on low-cardinality masks will lead to better generalization. Our ablation study supports this hypothesis, as cardinality reweighting indeed improves performance (Fig. 3).

Parallel training To account for parallel training, we proposed to train on the mask distribution $D_{M,w}(e) \propto \sum_{i \in X \setminus e} D_{M,w}(i, e)$. However, each pass of $D_{M,w}(e)$ trains on all of $\{p(x_i|\boldsymbol{x}_e) : i \in X \setminus e\}$, which overtrains neighboring edges and leads to a minor distributional mismatch. Aligning this training distribution more closely may lead to improvements, but requires better techniques for parallelization and sampling of masks. We leave this for future investigation.

5.1 Related Work

The idea of training AO-ARMs as autoregressive models on all possible orderings was first introduced in NADE [36, 35], and has been seen more recently in models such as ARDM [13] for image / text / audio and ACE [34] for continuous domains. AO-ARMs are also closely related to non-autoregressive language models such as Bert and XLNet [41]. Our method, MAC, proposes improved training and inference techniques for AO-ARMs.

Other techniques have also been proposed for non-autoregressive generation. For example, INTRUS [8] models sequences as consecutive insertion operations, as opposed to AO-ARMs which model sequences as consecutive unmasking operations. This also enables arbitrary order of generation, but prevents INTRUS from computing marginal likelihoods on subsets of variables. Another work [43] casts the problem of non-autoregressive generation as a GFlowNet, taking on a reinforcement learning framework with states, actions, and rewards. However, this approach currently does not scale as well as AO-ARMs.

Lastly, some works have studied the problem of learning a single generation order [20]. Although learning orderings is not applicable to vanilla AO-ARMs (since they train on all possible orders uniformly), it is applicable to MAC, which uses decomposition protocols that can be learned. For example, we can learn different canonical orderings of the variable indices in the binary lattice. Though we did not explore this direction thoroughly, we noticed that a strided canonical ordering (0, 32, 64, ..., 1, 33, ...) often did better than the standard lexicographical ordering for images, suggesting promise for learning even better canonical orderings, or learning better protocols in general.

5.2 Limitations

In practice, likelihood evaluations of standard AO-ARMs are done by sampling a finite number of orders (typically just a single order), leading to unbiased but approximate estimates of joint likelihoods. Moreover, evaluating marginal $p(\boldsymbol{x}_e)$ and conditional likelihoods $p(\boldsymbol{x}_u|\boldsymbol{x}_e)$ is trickier and has additional tradeoffs. Since AO-ARMs evaluate marginal and conditional likelihoods using only compatible orderings, they are implicitly discarding away all the models in the $N!$ ensemble with incompatible orderings. This can give biased estimates for marginal and conditional likelihoods.

For conditional likelihoods specifically, there are two potential methods for evaluation. First, we can evaluate $\log p(\boldsymbol{x}_u|\boldsymbol{x}_e)$ as $\log p(\boldsymbol{x}_u\boldsymbol{x}_e) - \log p(\boldsymbol{x}_e)$ and estimate two marginal likelihoods. However, since the marginal likelihoods are approximate, this can lead to invalid conditional probability estimates where $p(\boldsymbol{x}_u|\boldsymbol{x}_e) > 1$ (for discrete domains). Second, we can decompose $\boldsymbol{x}_u$ directly and evaluate $\log p(\boldsymbol{x}_u|\boldsymbol{x}_e)$ as $\sum_{i=1}^{|u|} \log p(\boldsymbol{x}_{u_{:i}}|\boldsymbol{x}_{u_{i:}}\boldsymbol{x}_e)$, where the ordering of variables within $\boldsymbol{x}_u$ is sampled randomly. This corresponds to evaluating random path that goes from node e to node $u \cup e$ in the binary lattice. Though this leads to valid probability estimates, the estimates may be more biased because paths that go to node $u \cup e$ without passing through node e are not considered.

MAC shares similar limitations as standard AO-ARMs, where marginal likelihoods estimates may be biased. Evaluating conditional likelihoods as two marginal likelihoods can lead to invalid (greater than 1) estimates, and evaluating them by tracing paths through the lattice from node e to node $u \cup e$ may be more biased.

Nonetheless, both marginal and conditional likelihood estimates (using Method 2) for MAC and standard AO-ARMs *are valid* in the sense that $\sum_{\boldsymbol{x}_u} p(\boldsymbol{x}_u) = 1$ and $\sum_{\boldsymbol{x}_u} p(\boldsymbol{x}_u|\boldsymbol{x}_e) = 1$ for any u and e. Training MAC to optimize for marginal likelihoods (Table 1/2/3) or for both marginal/conditional likelihoods (Table 4/5) still gives good empirical performance despite these limitations.

6 Conclusion

We present MAC, an improved method of training and inference on AO-ARMs. MAC trains on a carefully designed distribution of univariate conditionals that **(A)** reduces modeling redundancy of AO-ARMs and **(B)** aligns the training objective of AO-ARMs with arbitrary conditional queries. Our

method leads to better joint and arbitrary conditional likelihoods with no sacrifices on tractability. We show state-of-the-art results in arbitrary conditional modeling on text, image, continuous data domains, and present a novel application of arbitrary conditional models to robot shared-autonomy.

References

[1] Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems 34*, 2021.

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33*, 2020.

[3] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *CoRR*, abs/1904.10509, 2019.

[4] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the CIFAR datasets. *CoRR*, abs/1707.08819, 2017.

[5] Adnan Darwiche. A differential approach to inference in bayesian networks. *J. ACM*, 50(3):280–305, 2003.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR*, 2009.

[7] Anca D. Dragan and Siddhartha S. Srinivasa. A policy-blending formalism for shared control. *The International Journal of Robotics Research*, 2013.

[8] Dmitrii Emelianenko, Elena Voita, and Pavel Serdyukov. Sequence modeling with unconstrained generation order. *Advances in Neural Information Processing Systems*, 32, 2019.

[9] Pete Florence, Corey Lynch, Andy Zeng, Oscar A. Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. In *Conference on Robot Learning*, 2021.

[10] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020.

[11] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In *3rd Annual Conference on Robot Learning*, 2019.

[12] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems 33*, 2020.

[13] Emiel Hoogeboom, Alexey A. Gritsenko, Jasmijn Bastings, Ben Poole, Rianne van den Berg, and Tim Salimans. Autoregressive diffusion models. In *10th International Conference on Learning Representations*, 2022.

[14] Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 1999.

[15] Heewoo Jun, Rewon Child, Mark Chen, John Schulman, Aditya Ramesh, Alec Radford, and Ilya Sutskever. Distribution augmentation for generative modeling. In *Proceedings of the 37th International Conference on Machine Learning, ICML*, 2020.

[16] Diederik P Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models. In *Advances in Neural Information Processing Systems 34*, 2021.

[17] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[18] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[19] Alexander Ku, Peter Anderson, Roma Patel, Eugene Ie, and Jason Baldridge. Room-across-room: Multilingual vision-and-language navigation with dense spatiotemporal grounding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP*, 2020.

[20] Xuanlin Li, Brandon Trabucco, Dong Huk Park, Michael Luo, Sheng Shen, Trevor Darrell, and Yang Gao. Discovering non-monotonic autoregressive orderings with variational inference. In *9th International Conference on Learning Representations, ICLR 2021*, 2021.

[21] Yang Li, Shoaib Akbar, and Junier Oliva. Acflow: Flow models for arbitrary conditional likelihoods. In *Proceedings of the 37th International Conference on Machine Learning ICML*, 2020.

[22] Dylan P. Losey, Krishnan Srinivasan, Ajay Mandlekar, Animesh Garg, and Dorsa Sadigh. Controlling assistive robots with learned latent actions. In *IEEE International Conference on Robotics and Automation, ICRA*, 2020.

[23] Matt Mahoney. Large text compression benchmark, 2011.

[24] Alejandro Molina, Antonio Vergari, Karl Stelzner, Robert Peharz, Pranav Subramani, Nicola Di Mauro, Pascal Poupart, and Kristian Kersting. Spflow: An easy and extensible library for deep probabilistic learning using sum-product networks. *CoRR*, abs/1901.03704, 2019.

[25] Kevin P. Murphy, Yair Weiss, and Michael I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence UAI*, 1999.

[26] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, 2018.

[27] Judea Pearl. Reverend bayes on inference engines: A distributed hierarchical approach. In *Proceedings of the National Conference on Artificial Intelligence*, 1982.

[28] Dean Pomerleau. ALVINN: an autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems 1, NIPS*, 1988.

[29] Hoifung Poon and Pedro M. Domingos. Sum-product networks: A new deep architecture. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence UAI*, 2011.

[30] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *Proceedings of the 38th International Conference on Machine Learning ICML*, 2021.

[31] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P. Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. In *5th International Conference on Learning Representations, ICLR*, 2017.

[32] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *Proceedings of the 32nd International Conference on Machine Learning, ICML*, 2015.

[33] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In *Advances in Neural Information Processing Systems 32*, 2019.

[34] Ryan R. Strauss and Junier B. Oliva. Arbitrary conditional distributions with energy. In *Advances in Neural Information Processing Systems 34*, 2021.

[35] Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation. *The Journal of Machine Learning Research*, 17(1):7184–7220, 2016.

[36] Benigno Uria, Iain Murray, and Hugo Larochelle. A deep and tractable density estimator. In *Proceedings of the 31th International Conference on Machine Learning*, 2014.

[37] Aäron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Koray Kavukcuoglu, Oriol Vinyals, and Alex Graves. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems 29*, 2016.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, 2017.

[39] Martin J. Wainwright and Michael I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 2008.

[40] Mike Wu and Noah D. Goodman. Multimodal generative models for scalable weakly-supervised learning. In *Advances in Neural Information Processing Systems 31*, 2018.

[41] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. XLNet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems 32*, 2019.

[42] Raymond A. Yeh, Chen Chen, Teck-Yian Lim, Alexander G. Schwing, Mark Hasegawa-Johnson, and Minh N. Do. Semantic image inpainting with deep generative models. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2017.

[43] Dinghuai Zhang, Nikolay Malkin, Zhen Liu, Alexandra Volokhova, Aaron C. Courville, and Yoshua Bengio. Generative flow networks for discrete probabilistic modeling. In *International Conference on Machine Learning, ICML 2022*, 2022.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] In the supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] Due to computational limitations we ran on one seed.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] All datasets are publicly available.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Connections between Mask Distributions and Variable Ordering

We draw connections between our proposed framework over mask distributions and prior works' formulation over variable order distributions. In our framework, the decomposition protocol w specifies the probability of reducing a mask into one of its parents. By recursively applying w, we can trace a path on the binary lattice from a mask e to the root node $\emptyset$. This path corresponds to a (partial) variable order compatible with e. Under this interpretation, since our proposed protocol w-MAC is deterministic, it use a deterministic variable ordering (that is different) for each mask.

We can treat the variable order σ as a random variable and view the likelihood as a variational objective, as is done in prior work [36, 13]. Previous AO-ARMs, which sample ordering at random, use a uniform prior $\mathcal{U}(\sigma)$ over orderings, leading to the following ELBO.

$$\begin{aligned}\log p(\boldsymbol{x}) &= \log \sum_{\sigma} p(\sigma) p(\boldsymbol{x}|\sigma) = \log \sum_{\sigma} \mathcal{U}(\sigma) \prod_{t=1}^{N} p(x_{\sigma(t)}|\boldsymbol{x}_{\sigma(<t)};\sigma) \\ &\geq \sum_{\sigma} \mathcal{U}(\sigma) \sum_{t=1}^{N} \log p(x_{\sigma(t)}|\boldsymbol{x}_{\sigma(<t)};\sigma) = \mathbb{E}_{\sigma\sim\mathcal{U}(\sigma)} \sum_{t=1}^{N} \log p(x_{\sigma(t)}|\boldsymbol{x}_{\sigma(<t)};\sigma)\end{aligned}$$

Instead our proposed method uses a delta prior $w(e)$ over orderings (different for each mask), as induced by the protocol w-MAC. Due to the delta prior, there is no ELBO gap when using deterministic ordering functions.

$$\begin{aligned}\log p(\boldsymbol{x}_e) &= \log \sum_{\sigma} p(\sigma) p(\boldsymbol{x}_e|\sigma) = \log \sum_{\sigma} \mathbb{1}_{\sigma=w(e)} \prod_{t=1}^{|e|} p(x_{\sigma(t)}|\boldsymbol{x}_{\sigma(<t)};\sigma) \\ &= \log \left[\prod_{t=1}^{|e|} p(x_{\sigma(t)}|\boldsymbol{x}_{\sigma(<t)};\sigma)\right]_{\sigma=w(e)} = \left[\sum_{t=1}^{|e|} \log p(x_{\sigma(t)}|\boldsymbol{x}_{\sigma(<t)};\sigma)\right]_{\sigma=w(e)}\end{aligned}$$

Although this connection to variable ordering is interesting, we note that deterministic orderings do not fully capture the subtleties of mask distributions. In particular, our decomposition protocol was chosen to increase concentration of intermediate masks (lower entropy), which corresponds to picking variable orderings that are not just deterministic, but that overlap in paths on the lattice.

B Batch Sampling from MAC's Mask Distribution

Sampling masks from the training distribution *in batch* is important for efficient training. Since MAC uses a simple protocol (always decompose by removing the greatest element in the mask), we can sample masks in batch without simulating the decomposition through the lattice. We provide code snippets for this batch sampling procedure in PyTorch.

```
from torch import arange, multinomial, rand, randint

def sample_test_masks(batch: int, xdim: int):
    sigma = rand(size=(batch, xdim)).argsort(dim=-1)
    t = randint(low=1, high=xdim+1, size=(batch, 1))
    masks = sigma < t
    return masks, t

def sample_train_masks(batch: int, xdim: int):
    test_masks, test_t = sample_test_masks(batch, xdim)

    # sample intermediate prefix by taking random int in [0, test_t)
    batch_arange = arange(xdim).reshape(1, xdim).repeat(batch, 1)
```

```
    nonzero_weights = (batch_arange < test_t).float()
    t = multinomial(nonzero_weights, num_samples=1)

    # double argsort trick to get ranks, but we need:
    # 1. descending=True to order 1s before 0s of the bitmask
    # 2. stable=True to keep the relative ordering between the 1s
    sigma = test_masks.long().sort(descending=True,
                                   stable=True).indices.argsort()
    masks = sigma < t
    return masks, t

def cardinality_reweighting(batch: int, xdim: int, outer_batch=100):
    train_masks, train_t = sample_train_masks(outer_batch*batch, xdim)
    card_weight = (train_t + 1).float()[:,0]
    idx_select = multinomial(card_weight,
                             num_samples=batch, replacement=False)
    return train_masks[idx_select]
```

The function `sample_test_masks` samples in batch from the test mask distribution $M = M_{\text{card-mask}}$, where we first sample a cardinality uniformly random, and then sample a mask with the chosen cardinality uniformly at random. This function is modular – any other test mask distribution M can be dropped in without modifying rest of the code snippet.

The next function `sample_train_masks` samples masks from the training distribution $D_{M,w\text{-MAC}}$ in batch. It takes each test mask e and samples an intermediate mask on the path between e and $\emptyset$ as dictated by the decomposition protocol w-MAC. Since our protocol always removes the greatest element in the set, we can actually sample this intermediate mask without simulating the protocol: sort the elements and choose a prefix with length picked uniformly at random.

Finally, the function `cardinality_reweighting` changes the training distribution from $D_{M,w\text{-MAC}}$ to $D^{\text{CR}}_{M,w\text{-MAC}}$ using the cardinality reweighting heuristic. This is done by Sampling-Importance-Resampling, where we sample an excess (e.g., 100x) number of masks, reweight by their cardinality, and resample using these weights.

To train MAC, we sample masks in batch from $D^{\text{CR}}_{M,w\text{-MAC}}$ by calling `cardinality_reweighting`.

C Samples

We show (uncurated) masked-conditional samples from MAC, by masking test data and using MAC to fill in the missing dimensions. We repeat this for Text8, CIFAR10, and ImageNet32.

C.1 Text8

Each snippet displays 3 chunks of 250 characters. The first chunk is an unmasked example from the test set. The second chunk is its masked out version, with underscores denoting missing dimensions. Finally, the last chunk is the mask-conditional sample, using MAC to fill in the missing dimensions.

```
be ejected and hold it there examine the chamber to ensure it is clear
 allow the action to go forward under control push the forward assist
fire the action and close the ejection port cover safety precaution ma
gazine fitted perform an unload if the a

be _j_c__d a__ h__d _t t__re exa_i_e_the __a__er to __sure_it __ c_ea_
 allo__the a__i__ __ g_ _o_war_ u_der __ntrol p__h _h_ f_rw_rd_ass_st
f_re ___ action _nd_clo_e the _jection p__t _o_er _a_ety _r_ca_ti_n m_
g__i__ _i__ed_p____rm_a_ unlo___if_th_ a

be ejected and hold it to re examine the bladder to ensure it is clear
 allow the action to go forward under control push the forward assist
fire the action and close the ejection part cover safety precaution ma
gazine firied perform an unload if the a
```

```
 prophylactic drugs several drugs most of which are also used for trea
tment of malaria can be taken preventatively generally these drugs are
 taken daily or weekly at a lower dose than would be used for treatmen
t of a person who had actually contracte

 pr_p_y__cti__dr__s _eve__l drug_ _o_t of__h__h are __so u_ed __r _re_
tment__f mal__i__c__ b_ _a_en _re_enta__vely ge__ra_l_ t_es_ d_ugs_are
 _ake___aily or _ee_ly _t a lo_er dose t__n _o_ld _e_used_f_r _re_tme_
t__f__ _e__on_w____ad_a_tuall___on_ra_te

 prophylactic drugs several drugs most of which are also used for trea
tment of malaria can be taken preventatively generally these drugs are
 taken daily or weekly at a lower dose than would be used for treatmen
t of a lesion who had actually contracte
```

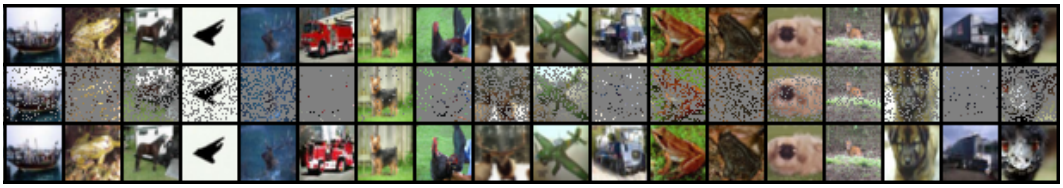
```
 themselves as a versified journal secondly there are cycles of poems
which fall into a regular chronological sequence among the single poem
s evidence that certain themes demanded further expression and develop
ment one cycle announces the theme of mi

____mse__e_______v_r__f__d__o_____ _____d_y______ ___ ____e_ o__p____
_h_c_ f__l_____ _ _______ ch___o___i_a__s_______ ___ng_th__si__l_ po_m
________c_ _h____er_ai_ ________e_____d __r____ _x__e______a_d_d_v____
_____o_e____l_ a__o_______he t_em_______

 themselves they vary foad loafing it a day three the number of poems
which follow at a at b is chronological stead up owing the single poem
 syllagics they certail a unprecedented form of expression and develop
ment one could also feel the theme can f
```

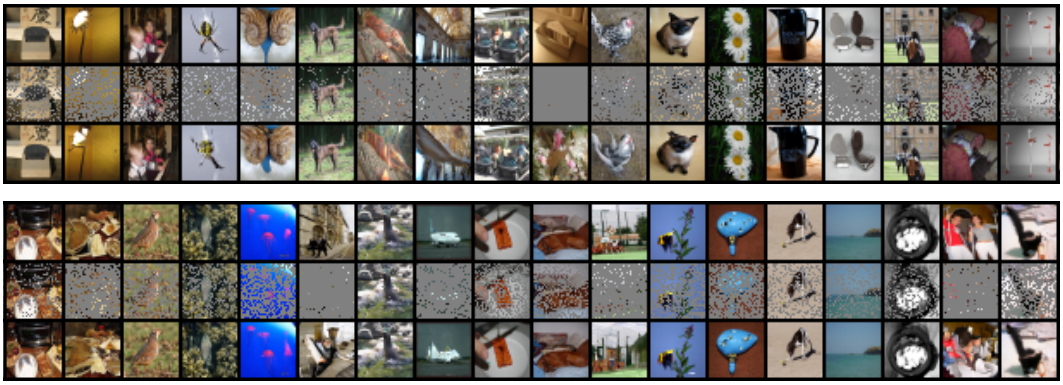
C.2 CIFAR10

Each figure displays 3 rows of 18 images. The first row is an unmasked example from the test set. The second row is its masked out version, with grey pixels denoting missing dimensions. Finally, the last chunk is the mask-conditional sample, using MAC to fill in the missing dimensions. For easier visualization, in these examples we align the masked dimensions for the three image channels (RGB). We do not align them during training and evaluation of test log-likelihoods, i.e., the masks could be different for each channel.



C.3 ImageNet32

We show samples from ImageNet32, using the same setup as described for CIFAR10 above.



D Experimental Details

For the language and image experiments, all of our architectural and hyperparameter choices are kept the same as ARDM [13], with the exception of the Cross-Entropy objective from [1]. We omit this Cross-Entropy objective, as the authors from [13] found "no substantial differences in performance" from including it. Compared to the ARDM baseline, we only modify the training mask distribution and inference protocol.

Language

- 12 layer Transformer with 768 dimensions, 12 heads, 3072 MLP dimensions, no dropout
- Batch size 512, chunk size 250 with *no additional context*
- Learning rate 5e-4, with linear warmup for the first 5k steps, using AdamW
- Gradient clipped at 0.25

Image

- U-Net with 32 ResBlocks at resolution 32×32 with 256 channels, interleaved with attention blocks, no dropout
- The mask is concatenated to the input, which is encoded using $3/4$ of the channels. The remaining $1/4$ of the channels encode the mask cardinality (see [13] for details).
- Batch size 128
- Learning rate 1e-4, with beta parameters (0.9 / 0.999), using Adam
- Gradient clipped at 100

Continuous Tabular Data For the continuous tabular data, all of our architectural and hyperparameter choices are kept the same as ACE [34]. Compared to the ACE baseline, we only modify the training mask distribution and inference protocol.

We use fully-connected residual architectures (with 4 residual blocks) for both proposal and energy networks. The proposal network uses a mixture of 10 Gaussian components. The exact configuration for each of the dataset is kept identical to ACE, and can be found in more detail in [34].

FrankaKitchen We train on the `Kitchen-mixed0` task, using a fully connected MLP with width 2048 and depth 8. We optimize using Adam with learning rate 5e-4, for 300 epochs with a batch size of 4k. BC learns an explicit model that directly maps a state to an action. MAC learns an implicit model that models a distribution over the action space given the state, as in [9]. The action distribution is modeled by univariate conditionals parameterized by a mixture of 20 Gaussians.