DISCOVERING AUTOREGRESSIVE ORDERINGS WITH VARIATIONAL INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

The predominant approach for language modeling is to encode a sequence of tokens from left to right, but this eliminates a source of information: the order by which the sequence was naturally generated. One strategy to recover this information is to decode both the *content* and *ordering* of tokens. Prior work supervises content and ordering with hand-designed loss functions or bootstraps from a predefined ordering. These approaches require domain-specific insight. We address this limitation with an unsupervised learner that discovers high-quality autoregressive orders without a domain-specific prior. The learner is a neural network that performs variational inference with the autoregressive order as a latent variable. Since the corresponding variational lower bound is not differentiable, we develop a practical algorithm for end-to-end optimization using policy gradients. Strong empirical results with our solution on various sequence modeling tasks suggest that our algorithm is capable of discovering various autoregressive orders for different sequences that are competitive with or even better than fixed orders.

1 INTRODUCTION

Autoregressive models have a rich history. Early papers that studied autoregressive models, such as (Uria et al., 2016) and (Germain et al., 2015), showed an interest in designing algorithms that did not require a gold-standard autoregressive order to be known upfront by researchers. However, these papers were overshadowed by developments in natural language processing that demonstrated the power of the left-to-right autoregressive order (Cho et al., 2014; Sutskever et al., 2014a). Since then, the left-to-right autoregressive order has been essential for application domains such as image captioning (Vinyals et al., 2015b; Xu et al., 2015), machine translation (Luong et al., 2015; Bahdanau et al., 2015) and distant fields like image synthesis (van den Oord et al., 2016). However, interest in non left-to-right autoregressive orders is resurfacing (Welleck et al., 2019b; Stern et al., 2019), and evidence (Vinyals et al., 2016; Gū et al., 2018; Alvarez-Melis & Jaakkola, 2017) suggests adaptive orders may produce more accurate autoregressive models. These positive results make designing algorithms that can leverage adaptive orders an important research domain.

Inferring autoregressive orderings in a data-driven manner is challenging. Modern benchmarks for machine translation (Stahlberg, 2019) and other tasks (Oda et al., 2015) are not labelled with gold-standard orders, and left-to-right seems to be the default. This could be explained if domain-independent methodology for identifying *high-quality* orders is an open question. Certain approaches (Stern et al., 2019; Welleck et al., 2019b; Ruis et al., 2020) use hand-designed loss functions to promote a *genre* of orders—such as balanced binary trees. These loss functions incorporate certain domain-assumptions: for example, they assume the balanced binary tree order will not disrupt learning. Learning disruption is an important consideration, because prior work shows that poor orders may prohibitively slow learning (Chen et al., 2018). The modeling assumptions described above lack domain-independence. *Ideal* frameworks for inferring autoregressive orders should withhold domain knowledge, making them general-purpose.

To our best knowledge, we propose the first domain-independent unsupervised learner that discovers high-quality autoregressive orders through fully-parallelizable end-to-end training without domain-specific tuning. We provide three main contributions that stabilize this learner. First, we propose an encoder architecture that conditions on training examples to output autoregressive orders using techniques in combinatorical optimization. Second, we propose a procedure named *Variational Order*

Inference that uses the architecture as a variational distribution and learns an approximate posterior over autoregressive orders. Finally, we develop a practical algorithm for solving the resulting non-differentiable ELBO end-to-end with policy gradients.

Empirical results with our solution on image captioning, code generation, and text summarization tasks suggest that with similar hyperparameters, our algorithm is capable of recovering autoregressive orders that are even better than fixed orders. Case studies suggest that our learned orders depend adaptively on content, and resemble a type of *best-first* generation order, which first decodes focal objects and names. Our experimental framework is made available at this link.

2 Related Works

Autoregressive Models Autoregressive models decompose the generation of a high dimensional probability distribution by generating one dimension at a time, with a predefined order. Combined with high capacity neural networks, this approach to modeling complex distributions has been very successful (Sutskever et al., 2011; Mikolov et al., 2012). Recent works have achieved great improvements with autoregressive models in many applications, including language modeling (Radford et al., 2018; 2019; Brown et al., 2020), machine translation (Sutskever et al., 2014b) and image captioning (Karpathy & Fei-Fei, 2015). Most previous works on autoregressive models use a fixed ordering pre-defined by the designer with left-to-right emerging as the primary choice. In contrast, our method is capable of learning arbitrary ordering conditional on data and is thus more flexible.

Non-Sequential Autoregressive Orderings Ford et al. (2018b) shows that a sub-optimal ordering can severely limit the viability of a language model and propose to first generate a partially filled sentence template and then fill in missing tokens. Previous works have also studied bidirectional decoding (Sun et al., 2017; Zhou et al., 2019; Mehri & Sigal, 2018) and syntax trees based decoding (Yamada & Knight, 2001; Charniak et al., 2003; Dyer et al., 2016; Aharoni & Goldberg, 2017; Wang et al., 2018) in the natural language setting. However, all of the works mentioned above do not learn the orderings and instead opt to use heuristics to define them. Chan et al. (2019) performs language modeling according to a known prior, such as balanced binary tree, and does not allow arbitrary sequence generation orders. Welleck et al. (2019a) proposes to use a tree-based recursive generation method to learn arbitrary generation orders. However, their performance lags behind that of left-to-right. Gu et al. (2019a) proposes Transformer-InDIGO to allow sequence generation through insertion operations. They first pretrain the model on pre-defined orderings, such as left-to-right, then use Searched Adaptive Order (SAO) to train on beam-searched alternative orderings. However, they report that without such pretraining, the model learns degenerate orderings. In addition, SAO requires keeping a beam of orderings at each time step, so training cannot be efficiently parallelized. Emelianenko et al. (2019) proposes to re-apply Transformer (Vaswani et al., 2017) self-attention at each time step to generate the next token with its position, but they also suffer from suboptimal theoretical training complexity. In contrast, our method is capable of learning high-quality autoregressive orders directly from data under fully-parallelizable end-to-end training without utilizing pre-defined orders or relying on domain knowledge.

Variational Methods Our method optimizes the evidence lower bound, or ELBO in short. ELBO is a quantity that is widely used as an optimization proxy in the machine learning literature, where the exact quantity is hard to compute or optimize. Variational methods have achieved great success in machine learning, such as VAE (Kingma & Welling, 2013) and β-VAE (Higgins et al., 2017).

Combinatorical Optimization Recent works have proposed various methods for gradient-based optimization in the large discrete space of permutations (Mena et al., 2018; Grover et al., 2019; Linderman et al., 2018). These works have applied such techniques in tasks such as number sorting, jigsaw puzzle solving, and neural signal identification in worms. To our best knowledge, we are the first to build on these techniques to discover autoregressive orderings in vision and language datasets such as image captioning and code generation.

3 Preliminaries

The goal of autoregressive sequence modelling is to model an ordered sequence of target values $\mathbf{y} = (y_1, y_2 \ldots, y_n) : y_i \in \mathbb{R}$, possibly conditioned on an ordered sequence of source values $\mathbf{x} = (x_1, x_2 \ldots, x_m) : x_i \in \mathbb{R}$, where $(\mathbf{x}, \mathbf{y})$ is sampled from the dataset $\mathcal{D}$.

Inspired by Vinyals et al. (2015a) and Gu et al. (2019a), we formulate the generation process of $\mathbf{y}$ as a $2n$ step process, where at time step $2t-1$ we generate a value, and at timestep $2t$ we select a not-yet-chosen position in $\{1, 2, \cdots, n\}$ to insert the value. Thus, we introduce the latent sequence variable $\mathbf{z} = (z_1, z_2 \ldots, z_n) : \mathbf{z} \in S_n$, where S_n is the set of one-dimensional permutations of $\{1, 2, \cdots, n\}$, and z_t is defined as the absolute position of the value generated at time step $2t-1$ in the naturally ordered $\mathbf{y}$. Then $p(\mathbf{y}, \mathbf{z}|\mathbf{x})$ denotes the probability of generating $\mathbf{y}$ in the ordering of $\mathbf{z}$ given the source sequence $\mathbf{x}$. We can thus factorize $p(\mathbf{y}, \mathbf{z}|\mathbf{x})$ using the chain rule:

$$p(\mathbf{y}, \mathbf{z}|\mathbf{x}) = p(y_{z_1}|\mathbf{x})p(z_1|y_{z_1}, \mathbf{x}) \prod_{i=2}^{n} p(y_{z_i}|z_{<i}, y_{z_{<i}}, \mathbf{x})p(z_i|z_{<i}, y_{z_{<=i}}, \mathbf{x}) \tag{1}$$

For example, $p(y_1, y_2, z_1 = 2, z_2 = 1|\mathbf{x}) = p(y_2|\mathbf{x})p(z_1|y_2, \mathbf{x})p(y_1|z_1, y_2, \mathbf{x})p(z_2|y_1, z_1, y_2, \mathbf{x})$ is defined as the probability of generating y_2 in the first step, then inserting y_2 into position 2, then generating y_1, and finally inserting y_1 into position 1.

Note that in practice, the length of $\mathbf{y}$ is usually varied. Therefore, we do not first create a fixed-length sequence of blanks and then replace the blanks with actual values. Instead, we dynamically insert a new value at a position relative to the previous values. One common approach to predict such relative position is Pointer Network (Vinyals et al., 2015a). In other words, at timestep t, we insert the value at position r_t relative to the previous generated values. Here, for any $\mathbf{z} \in S_n$, $\mathbf{r} = (r_1, r_2, \ldots, r_n)$ is constructed such that there is a bijection between S_n and the set of all constructed $\mathbf{r}$. Due to such bijection, we are able to use $\mathbf{z}$ and $\mathbf{r}$ exchangeably. For convenience, we use $\mathbf{z}$ throughout the paper.

4 Variational Order Inference (VOI)

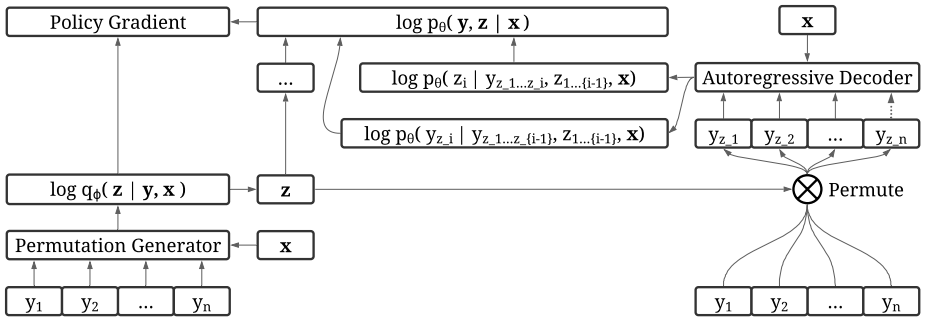


Figure 1: Computational Graph for Variational Order Inference

Starting from just the original data $\mathbf{y}$ in natural order, we can use variational inference to create an objective (2) that allows us to recover latent order $\mathbf{z}$, parametrized by two neural networks θ and ϕ. The encoder network ϕ samples autoregressive orders given the ground truth data, which the decoder network θ uses to recover $\mathbf{y}$. More specifically, ϕ is a non-autoregressive network (permutation generator in Fig. 1) that takes in the source sequence $\mathbf{x}$ and the entire ground truth target sequence $\mathbf{y}$ and outputs latent order $\mathbf{z}$ in a single forward pass. θ is an autoregressive network (autoregressive decoder in Fig. 1) that takes in $\mathbf{x}$ and predicts both the target sequence $\mathbf{y}$ and the ordering $\mathbf{z}$ through the factorization in Equation (1). We name this process *Variational Order Inference* (VOI).

$$\begin{aligned} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}}\left[\log p_\theta(\mathbf{y}|\mathbf{x})\right] &= \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}}\left[\log \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{y},\mathbf{x})}\left[\frac{p_\theta(\mathbf{y},\mathbf{z}|\mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{y},\mathbf{x})}\right]\right] \\ &\geq \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}}\left[\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{y},\mathbf{x})}\left[\log p_\theta(\mathbf{y},\mathbf{z}|\mathbf{x})\right] + \mathcal{H}_{q_\phi}(\cdot|\mathbf{y},\mathbf{x})\right] \end{aligned} \tag{2}$$

Here, $\mathcal{H}_{q_\phi}$ is the entropy term. During training, we train ϕ and θ jointly to maximize the ELBO in (2). During testing, we only keep the decoder θ.

To optimize the decoder network θ in (2), for each $\mathbf{y}$, we first sample K latents $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_K$ from $q_\phi(\cdot|\mathbf{y}, \mathbf{x})$. We then update θ using the Monte-Carlo gradient estimate $\mathbb{E}_{\mathbf{y}\sim\mathcal{D}}\left[\frac{1}{K}\sum_{i=1}^{K} \nabla_\theta \log p_\theta(\mathbf{y}, \mathbf{z}_i|\mathbf{x})\right]$.

Algorithm 1 Variational Order Inference

Given: encoder network ϕ with learning rate α_ϕ, decoder network θ with learning rate α_θ, entropy coefficient β, batch of training data $(\mathbf{X}, \mathbf{Y}) = \{(\mathbf{x}_b, \mathbf{y}_b)\}_{b=1}^N$ sampled from dataset $\mathcal{D}$
Set gradient accumulators $g_\phi = 0$, $g_\theta = 0$
for $(\mathbf{x}, \mathbf{y}) \in (\mathbf{X}, \mathbf{Y})$ **do** ▷ In practice, this is done through parallel tensor operations
  $X = \phi(\mathbf{y}, \mathbf{x})$
  Sample K doubly stochastic matrices $B_1, B_2, \ldots, B_K \in \mathcal{B}_{n\times n}$ from $\mathcal{G}.\mathcal{S}.(X, \tau)$
  Obtain $P_1, P_2, \ldots, P_K \in \mathcal{P}_{n\times n}$ from $B_1, B_2, \ldots, B_K$ using Hungarian Algorithm
  Obtain latents $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_K = f^{-1}_{\text{len}(\mathbf{y})}(P_1), f^{-1}_{\text{len}(\mathbf{y})}(P_2), \ldots, f^{-1}_{\text{len}(\mathbf{y})}(P_K)$
  $g_\theta = g_\theta + \frac{1}{N\cdot K}\sum_{i=1}^K \nabla_\theta \log p_\theta(\mathbf{y}, \mathbf{z}_i|\mathbf{x})$
  Calculate $\log q_\phi(\mathbf{z}_i|\mathbf{y}, \mathbf{x}) = \langle X, P_i\rangle_F - \log(\text{perm}(\exp(X)))$
    $\approx \langle X, P_i\rangle_F - \log(\text{perm}_B(\exp(X)))$
  Calculate $b(\mathbf{y}, \mathbf{x}) = \frac{1}{K}\sum_{i=1}^K \log p_\theta(\mathbf{y}, \mathbf{z}_i|\mathbf{x})$
  $g_\phi = g_\phi + \frac{1}{N\cdot K}\sum_{i=1}^K \nabla_\phi \log q_\phi(\mathbf{z}_i|\mathbf{y}, \mathbf{x})(\log p_\theta(\mathbf{y}, \mathbf{z}_i|\mathbf{x}) - b(\mathbf{y}, \mathbf{x})) + \beta \cdot \nabla_\phi \mathcal{H}_{q_\phi}(\cdot|\mathbf{y}, \mathbf{x})$
end for
$\phi = \phi + \alpha_\phi \cdot g_\phi$
$\theta = \theta + \alpha_\theta \cdot g_\theta$

Optimizing the encoder network ϕ is tricky. Since $\mathbf{z}$ is a discrete latent variable, the gradient from $\log p_\theta(\mathbf{y}, \mathbf{z})$ does not flow through $\mathbf{z}$. Thus, we formulate (2) in a reinforcement learning setting with a one-step Markov Decision Process $(\mathcal{S}, \mathcal{A}, \mathcal{R})$. Under our setting, the state space $\mathcal{S} = \mathcal{D}$; for each state $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$, the action space $\mathcal{A}_{(\mathbf{x},\mathbf{y})} = S_{\text{length}(\mathbf{y})}$ with entropy term $\mathcal{H}_{q_\phi}(\cdot|\mathbf{y}, \mathbf{x})$; the reward function $\mathcal{R}((\mathbf{x}, \mathbf{y}), \mathbf{z} \in S_{\text{length}(\mathbf{y})}) = \log p_\theta(\mathbf{y}, \mathbf{z}|\mathbf{x})$. We can then set the optimization objective $L(\phi)$ to be (2). In practice, we find that adding an entropy coefficient β and gradually annealing it can speed up the convergence of decoder while still obtaining good autoregressive orders.

To compute $\nabla_\phi L(\phi)$, we derive the policy gradient with baseline formulation (Sutton et al., 2000):

$$\nabla_\phi L(\phi) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}}\left[\mathbb{E}_{\mathbf{z}\sim q_\phi}\left[\nabla_\phi \log q_\phi(\mathbf{z}|\mathbf{y}, \mathbf{x})(\log p_\theta(\mathbf{y}, \mathbf{z}|\mathbf{x}) - b(\mathbf{y}, \mathbf{x}))\right] + \beta\nabla_\phi \mathcal{H}_{q_\phi}\right] \tag{3}$$

where $b(\mathbf{y}, \mathbf{x})$ is the baseline function independent of action $\mathbf{z}$. The reason we use a state-dependent baseline $b(\mathbf{y}, \mathbf{x})$ instead of a global baseline b is that the the length of $\mathbf{y}$ can have a wide range, causing significant reward scale difference. In particular, we set $b(\mathbf{y}, \mathbf{x}) = \mathbb{E}_{\mathbf{z}\sim q_\phi}[\log p_\theta(\mathbf{y}, \mathbf{z}_i|\mathbf{x})]$. If we sample $K \geq 2$ latents for each $\mathbf{y}$, then we can use its Monte-Carlo estimate $\frac{1}{K}\sum_{i=1}^K \log p_\theta(\mathbf{y}, \mathbf{z}_i|\mathbf{x})$.

Since we use policy gradient to optimize ϕ, we still need a closed form for the distribution $q_\phi(\mathbf{z}|\mathbf{y}, \mathbf{x})$. Before we proceed, we define $\mathcal{P}_{n\times n}$ as the set of $n \times n$ permutation matrices, where exactly one entry in each row and column is 1 and all other entries are 0; $\mathcal{B}_{n\times n}$ as the set of $n \times n$ doubly stochastic matrices, i.e. non-negative matrices whose sum of entries in each row and in each column equals 1; $\mathbb{R}^+_{n\times n}$ as the set of non-negative $n \times n$ matrices. Note that we have the relationship $\mathcal{P}_{n\times n} \subset \mathcal{B}_{n\times n} \subset \mathbb{R}^+_{n\times n}$.

To obtain $q_\phi(\mathbf{z}|\mathbf{y}, \mathbf{x})$, we first write $\mathbf{z}$ in two-dimensional form. For each $\mathbf{z} \in S_n$, let $f_n(\mathbf{z}) \in \mathcal{P}_{n\times n}$ be constructed such that $f_n(\mathbf{z})_i = \text{one_hot}(z_i)$, where $f_n(\mathbf{z})_i$ is the i-th row of $f_n(\mathbf{z})$. Thus f_n is a natural bijection from S_n to $\mathcal{P}_{n\times n}$, and we can rewrite q_ϕ as a distribution over $\mathcal{P}_{n\times n}$ such that $q_\phi(f_n(\mathbf{z})|\mathbf{y}, \mathbf{x}) = q_\phi(\mathbf{z}|\mathbf{y}, \mathbf{x})$.

Next, we need to model the distribution of $q_\phi(\cdot|\mathbf{y}, \mathbf{x})$. Inspired by (Mena et al., 2018), we model $q_\phi(\cdot|\mathbf{y}, \mathbf{x})$ as a Gumbel-Matching distribution $\mathcal{G}.\mathcal{M}.(X)$ over $\mathcal{P}_{n\times n}$, where $X = \phi(\mathbf{y}, \mathbf{x}) \in \mathbb{R}^{n\times n}$ is the output of ϕ. Then for $P \in \mathcal{P}_{n\times n}$,

$$q_\phi(\mathbf{z}|\mathbf{y}, \mathbf{x}) = q_\phi(f_n^{-1}(P)|\mathbf{y}, \mathbf{x}) = q_\phi(P|\mathbf{y}, \mathbf{x}) \propto \exp\langle X, P\rangle_F \tag{4}$$

where $\langle X, P\rangle_F = \text{trace}(X^T P)$ is the Frobenius inner product of X and P. To obtain samples in $\mathcal{P}_{n\times n}$ from the Gumbel-Matching distribution, (Mena et al., 2018) relaxes $\mathcal{P}_{n\times n}$ to $\mathcal{B}_{n\times n}$ by defining the Gumbel-Sinkhorn distribution $\mathcal{G}.\mathcal{S}.(X, \tau) : \tau > 0$ over $\mathcal{B}_{n\times n}$, and proves that $\mathcal{G}.\mathcal{S}.(X, \tau)$ converges almost surely to $\mathcal{G}.\mathcal{M}.(X)$ as $\tau \to 0^+$. Therefore, to approximately sample from $\mathcal{G}.\mathcal{M}.(X)$, we first sample from $\mathcal{G}.\mathcal{S}.(X, \tau)$, then apply Hungarian algorithm (Munkres, 1957) to obtain $P \in \mathcal{G}.\mathcal{M}.(X)$. Further details are presented in Appendix A.

The Gumbel-Matching distribution allows us to obtain the numerator for the closed form of $q_\phi(\mathbf{z}|\mathbf{y}, \mathbf{x}) = q_\phi(f_n^{-1}(P)|\mathbf{y}, \mathbf{x})$, which equals $\exp\langle X, P\rangle_F$. However, the denominator is in-

tractable to compute and equals $\sum_{P \in \mathcal{P}_{n \times n}} \exp \langle X, P \rangle_F$. Upon further examination, we can express it as $\text{perm}(\exp(X))$, the matrix permanent of $\exp(X)$, and approximate it using $\text{perm}_B(\exp(X))$, its Bethe permanent. We present details about matrix permanent and Bethe permanent along with the proof that the denominator of $q_\phi(\cdot|\mathbf{y}, \mathbf{x})$ equals $\text{perm}(\exp(X))$ in Appendix B.

After we approximate q_ϕ, we can now optimize ϕ using the policy gradient in (3). We present a diagram of our architecture in Figure 1, and a pseudocode of our algorithm in Algorithm 1. Note that even though latent space S_n is very large and contains $n!$ permutations, in practice, if $p_\theta(\mathbf{y}, \mathbf{z}^*|\mathbf{x}) \geq p_\theta(\mathbf{y}, \mathbf{z}|\mathbf{x}) \ \forall \mathbf{z} \in S_n$, then $p_\theta(\mathbf{y}, \mathbf{z}|\mathbf{x})$ tends to increase as the edit distance between $\mathbf{z}$ and $\mathbf{z}^*$ decreases. Therefore, ϕ does not need to search over the entire latent to obtain good permutations, making variational inference over S_n feasible.

5 Experiments

Encoder and Decoder Architecture. We present encoder and decoder architectures for *Variational Order Inference* on conditional sequence generation tasks, which we focus on in this work. Note that Algorithm 1 is also applicable to unconditional sequence generation domains, such as image generation, through different encoder and decoder architectures. We leave this for future work.

For decoder θ, we use the Transformer-InDIGO (Gu et al., 2019a) architecture, which builds on Transformer with relative position representations (Shaw et al., 2018) to allow sequence generation through insertion operations. Note that "encoder" and "decoder" in this section refer to the two networks ϕ and θ in Algorithm 1, respectively, instead of Transformer's encoder and decoder. Also, we obtain orderings through the output of encoder instead of through Searched Adaptive Order (SAO). To our best effort, we were unable to obtain the official implementation of Transformer-InDIGO, so we reimplemented the algorithm based on the paper's descriptions.

As a side note, rather than generating $\mathbf{y}$ autoregressively in $2n$ steps as done in Transformer-InDIGO, it is possible to use a non-autoregressive decoder instead and improve the decoding speed. There have recently been many works on non-autoregressive conditional sequence generation (Gu et al., 2018; 2019b; Ma et al., 2019; Bao et al., 2019). To train a non-autoregressive decoder Transformer, we can incorporate the ordering information generated by our encoder network into the decoder Transformer's encoder latent output. We leave this for future work.

For encoder ϕ, we adopt the Transformer (Vaswani et al., 2017) architecture. Note that our encoder generates latents based on the entire ground truth target sequence $\mathbf{y}$. Therefore, it does not need to mask out subsequent positions during attention. We also experiment with different position embedding schemes (see Section 7) and find that Transformer-XL's (Dai et al., 2019) relative positional encoding performs the best, so we replace the sinusoid encoding in the original Transformer.

Tasks. We evaluate our approach on challenging sequence generation tasks: natural language to code generation (NL2Code) (Ling et al., 2016), image captioning, and text summarization. For NL2Code, we use Django (Oda et al., 2015). For image captioning, we use COCO 2017 (Lin et al., 2015). For text summarization, we use English Gigaword (Graff et al., 2003; Rush et al., 2015).

Baselines. We compare our approach with several pre-defined fixed orders: Left-to-Right (L2R) (Wu et al., 2018), Common-First (Common) (Ford et al., 2018a), Rare-First (Rare) (Ford et al., 2018a), and Random-Ordering (Random). Here, Common-First order is defined as generating words with ordering determined by their relative frequency from high to low; Rare-First order is defined as the reverse of Common-First order; and Random-Ordering is defined as training with a randomly sampled order for each sample at each time step.

Preprocessing. For Django, we adopt the same preprocessing steps as described in (Gu et al., 2019a), and we use all unique words as the vocabulary. For MS-COCO, we find that the baseline in Gu et al. (2019a) is much lower than commonly used in the vision and language community. Therefore, instead of using Resnet-18, we use the pretrained Faster-RCNN checkpoint using a ResNet-50 FPN backbone provided by TorchVision (Marcel & Rodriguez, 2010) to extract 512-dimensional feature vectors for each object detection. To make our model spatially-aware, we also concatenate the bounding box coordinates for every detection before feeding into our Transformer's encoder. For Gigaword, we learn 32k byte-pair encoding (BPE, Sennrich et al. (2016)) on tokenized data.

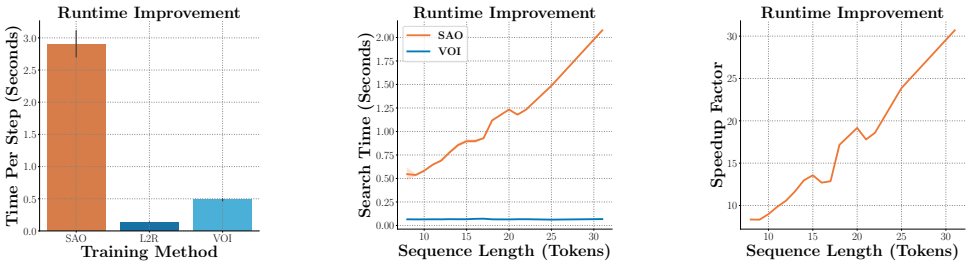


Figure 2: **Runtime performance improvement.** We compare the runtime performance of VOI ($K = 4$) with SAO on a single Tesla P100 GPU, in terms of time per training iteration and ordering search time. VOI outputs latent orderings in a single forward pass, and we observe a significant runtime improvement over SAO that searches orderings sequentially. The speedup factor linearly increases with respect to the sequence length.

Training. For our decoder, we set $d_{\text{model}} = 512$, $d_{\text{hidden}} = 2048$, 6 layers for both Transformer's encoder and decoder, and 8 attention heads. This is the same model configuration as Transformer-Base (Vaswani et al., 2017) and as described in Gu et al. (2019a). Our encoder also uses the same configuration. For all experiments, we apply dropout = 0.1 (Srivastava et al., 2014) and label smoothing = 0.1. For our model trained with *Variational Order Inference* , we sample $K = 4$ latents for each training sample. An ablation on choices of K is presented in Section 7. Detailed hyperparameters are listed in Appendix C.

During training, our encoder outputs the latent ordering through one single forward pass, and our decoder can predict all tokens with their positions given by the latent ordering in one single forward pass. If we let N denote the batch size, l denote the length of each target sequence, and d denote the size of hidden vector, then one single forward pass of our model has computation complexity $O(NKdl^2)$, while Transformer-InDIGO trained with SAO has complexity $O(Ndl^3)$. Since $K \ll l$ in general, our algorithm has better theoretical computational complexity during training. During evaluation, we only keep the decoder to iteratively generate the next position and token, which is as efficient as any standard fixed-order autoregressive models.

We also empirically compare VOI's runtime with that of SAO and fixed-order baselines (e.g. L2R). We implement SAO as described in Gu et al. (2019a). We test the runtime on a single GPU in order to accurately measure the number of ops required. For training speed per iteration, we use a batch size of 8. For ordering search time, we use a batch size of 1 to avoid padding tokens in the input for accurate measure. We observe that VOI is significantly faster than SAO, which searches orderings sequentially. In practice, as we distribute VOI across more GPUs, the K factor in the runtime is effectively divided by the number of GPUs used (if we ignore the parallelization overhead), so we can achieve further speedups.

Order	MS-COCO				Django		Gigaword		
	BLEU	Meteor	R-L	CIDEr	BLEU	Accuracy	R-1	R-2	R-L
InDIGO - L2R	22.12	—	—	68.88	36.74	13.60	—	—	—
InDIGO - SAO	22.58	—	—	69.42	42.33	16.30	—	—	—
Ours - Random	28.94	24.16	55.15	92.82	21.61	26.93	30.11	11.60	27.63
Ours - L2R	30.52	25.28	54.48	95.55	40.48	33.65	35.64	17.24	33.24
Ours - Common	28.02	24.75	55.52	90.27	37.13	29.82	33.89	15.01	31.10
Ours - Rare	28.11	24.48	52.85	91.42	31.08	27.93	34.11	15.15	31.26
Ours - VOI	**31.11**	**25.87**	**56.13**	**101.72**	**44.20**	**34.53**	**36.64**	**17.60**	**34.04**

Table 1: Results of MS-COCO, Django, and Gigaword with fixed orders (L2R, Random, Common, Rare) as baseline. We also report the performance of Transformer-INDIGO trained with L2R (baseline) and SAO from the original paper as reference ("—" = not reported). Here, R-1, R-2, and R-L indicate ROUGE-1, ROUGE-2, and ROUGE-L, respectively.

Results. We compare VOI against predefined orderings in Table 1. For MS-COCO, we use BLEU-4 (Papineni et al., 2002), Meteor (Denkowski & Lavie, 2014), Rouge (Lin, 2004), and CIDEr (Vedan-

tam et al., 2015) as metrics. For Django, besides reporting BLEU-4, we also report accuracy, defined as the percentage of perfect matches in code generation. For Gigaword, we use Rouge. Our results illustrate consistently better performance across fixed orderings. Most notably, CIDEr for MS-COCO, BLEU for Django, and Rouge-1 for Gigaword reveal the largest improvements in performance. Unfortunately, it is quite difficult to make a fair comparison between InDIGO and our method due to reproducibility issue regarding InDIGO. Specifically, the performance gap between InDIGO-L2R and Ours-L2R is large on Django, although they should be equivalent models. We thus mainly analyze our method in comparison to fixed-order baselines. Still, we report performance of InDIGO in the table for completeness.

6 Order Analysis

In this section, we analyze the generation orders learned by *Variational Order Inference* on a macro level by comparing the similarity of our learned orders with predefined orders defined in Section 5, and on a micro level, by inspecting when the model generates certain *types* of tokens.

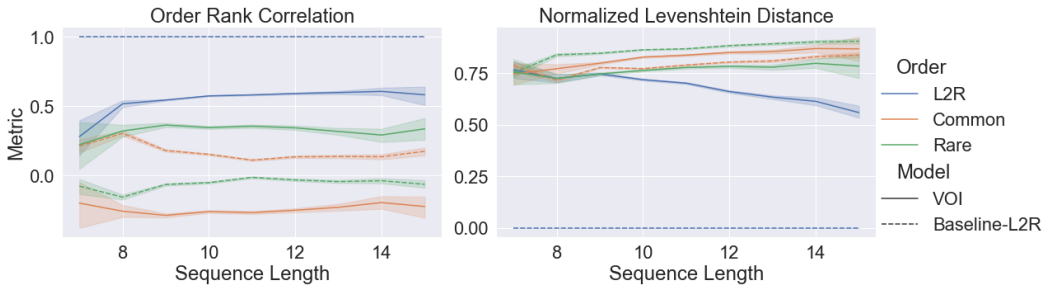


Figure 3: **Global statistics for learned orders.** We compare metrics as a function of the sequence length of generated captions on the COCO 2017 validation set. On the left, we compare orders learned with *Variational Order Inference* to a set of predefined orders (solid lines) using *Order Rank Correlation*. As a reference, we provide the *Order Rank Correlation* between L2R and the same set of predefined orders (dashed lines). In the right plot, with identical setup, we measure *Normalized Levenshtein Distance*. We observe that *Variational Order Inference* favors left-to-right decoding above the other predefined orders—this corresponds to the blue lines. However, with a max *Order Rank Correlation* of 0.6, it appears left-to-right is not a perfect explanation. The comparably high *Order Rank Correlation* of 0.3 with rare-tokens-first order suggests a complex strategy.

6.1 Understanding The Model Globally

We find that prior work (Gu et al., 2019a; Welleck et al., 2019a; Gu et al., 2018) tends to study autoregressive orders by evaluating performance on validation sets, and by visualizing the model's generation steps. We provide similar visualizations in Appendix F.3. However, this does not merit a quantitative understanding of the *strategy* that was learned. We address this limitation by introducing methodology to quantitatively study decoding strategies learned by non-monotonic autoregressive models. We introduce *Normalized Levenshtein Distance* and *Order Rank Correlation*, to measure similarity between decoding strategies. Given two generation orders $\mathbf{w}, \mathbf{z} \in S_n$ of the same sequence $\mathbf{y}$, where n is the length of $\mathbf{y}$, we define the *Normalized Levenshtein Distance*.

$$\mathcal{D}_{NLD}(\mathbf{w}, \mathbf{z}) = \text{lev}(\mathbf{w}, \mathbf{z}) / n \tag{5}$$

$$\text{lev}(\mathbf{w}, \mathbf{z}) = 1 + \min\{\text{lev}(\mathbf{w}_{1:}, \mathbf{z}), \text{lev}(\mathbf{w}, \mathbf{z}_{1:}), \text{lev}(\mathbf{w}_{1:}, \mathbf{z}_{1:})\} \tag{6}$$

The function $\text{lev}(\mathbf{w}, \mathbf{z})$ is the Levenshtein distance, and $z_{1:}$ removes the first element of z. This metric has the property that a distance of 0 implies that two orders $\mathbf{w}$ and $\mathbf{z}$ are the same, while a distance of 1 implies that the same tokens appear in distant locations in $\mathbf{w}$ and $\mathbf{z}$. Our second metric *Order Rank Correlation*, is the Spearman's rank correlation coefficient between $\mathbf{w}$ and $\mathbf{z}$.

$$\mathcal{D}_{ORC}(\mathbf{w}, \mathbf{z}) = 1 - 6 \cdot \sum_{i=0}^{n} (\mathbf{w}_i - \mathbf{z}_i) / (n^3 - n) \tag{7}$$

A correlation of 1 implies that $\mathbf{w}$ and $\mathbf{z}$ are the same; a correlation of -1 implies that $\mathbf{w}$ and $\mathbf{z}$ are reversed; and a correlation of 0 implies that $\mathbf{w}$ and $\mathbf{z}$ are not correlated. In Figure 3, we apply these metrics to analyze our models learnt through *Variational Order Inference* .

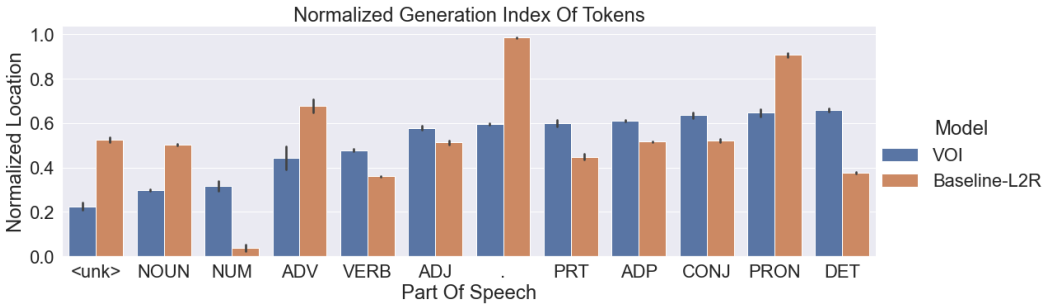


Figure 4: **Local statistics for learned orders.** In this figure, we evaluate the normalized generation indices for different parts of speech in model-predicted captions on the COCO 2017 validation set. The normalized generation index is defined as the absolute generation index of a particular token, divided by the final length of predicted sequence. The parts of speech (details in Appendix E) are sorted in ascending order of their average normalized location. We observe that *modifier* tokens, such as "the", tend to be decoded last, while *descriptive* tokens, such as nouns and verbs, tend to be decoded first.

Discussion. The experiment in Figure 3 confirms our model's behavior is not well explained by predefined orders. Interestingly, as the generated sequences increase in length, the *Normalized Levenshtein Distance* decreases, reaching a final value of 0.57, indicating that approximately half of the tokens are already arranged according to a left-to-right generation order. However, the *Order Rank Correlation* barely increases, so we can infer that while individual tokens are close to their left-to-right generation index, their relative ordering is not preserved. Our hypothesis is that certain phrases are generated from left-to-right, but their arrangement follows a *best-first* strategy.

6.2 Understanding The Model Locally

To complement the study of our model at a global level, we perform a similar study on the micro token level. Our hope is that a per-token metric can help us understand if and when our *Variational Order Inference* is adaptively choosing between left-to-right and rare-first order. We also hope to evaluate our hypothesis that *Variational Order Inference* is following a *best-first* strategy.

Discussion. The experiment in Figure 4 demonstrates that *Variational Order Inference* prefers decoding *descriptive* tokens first—such as nouns, numerals, adverbs, verbs, and adjectives. In addition, the unknown part of speech is typically decoded first, and we find this typically corresponds to special tokens such as proper names. Our model appears to capture the *salient* content first, which is illustrated by nouns ranking second in the generation order statistics. For image captioning, nouns typically correspond to focal objects, which suggests our model has an object-detection phase. Evidence of this phase supports our previous hypothesis that a *best-first* strategy is learned.

6.3 Understanding The Model Via Perturbations

In this section, we study the question: to what extent is the generation order learned by *Variational Order Inference* dependent on the content of the conditioning variable $\mathbf{x}$? This question is important because simply knowing that our model has learned a *best-first* does not illuminate whether that strategy depends only on the target tokens $\mathbf{y}$ being generated, or if it also depends on the content of $\mathbf{x}$. An adaptive generation order should depend on both.

Discussion. In this experiment, we first obtain a sequence $\mathbf{y}$ generated by our VOI given the source image $\mathbf{x}$. We then freeze $\mathbf{y}$, which allows the model to infer a new generation order for $\mathbf{y}$ when

different features of $\mathbf{x}$ are removed. Figure 6.3 shows that for a particular case, removing a single region-feature (feature number 0, which corresponds to the bus) from $\mathbf{x}$ changes the model-predicted generation order by as much as 0.7 *Normalized Levenshtein Distance*. These results confirm that our model appears to learn an *adaptive* strategy, which depends on both the tokens $\mathbf{y}$ being generated and the content of the conditioning variable $\mathbf{x}$, which is an image in this experiment.

7 Ablation Studies

In Section 5, we introduced the specific encoder and decoder architectures we use for conditional sequence generation tasks. In this section, we present ablation studies to support the architecture design of our encoder and modeling q_ϕ with Gumbel-Matching distribution.

Table 2: Normalized Levenshtein Distance between the ordering learnt by the encoder and the ground truth ordering, under different positional encodings (enc) and modeling distributions of q_ϕ (distrib).

Enc \ Distrib	Gumbel-Matching	Plackett-Luce
Sinusoid	0.40	0.62
Sinusoid + Pos Attn	0.42	0.58
Relative	0.38	0.53
XL-Relative	**0.25**	0.57

We consider 4 different positional encoding schemes for the encoder Transformer ϕ: the sinusoid encoding in the original Transformer (Vaswani et al., 2017), the sinusoid encoding with positional attention module (Gu et al., 2018), the relative positional encoding in Shaw et al. (2018), and the relative positional encoding proposed in Transformer-XL (Dai et al., 2019). Besides modeling $q_\phi(\cdot|\mathbf{x}, \mathbf{y})$ as Gumbel-Matching distribution and using Bethe permanent to approximate its denominator, we also consider modeling using Plackett-Luce distribution (Plackett, 1975; Luce, 1959) and sample using techniques recently proposed in Grover et al. (2019). Plackett-Luce distribution has tractable density, so we can compute the exact q_ϕ efficiently without using approximation techniques.

To analyze the encoder's ability to learn autoregressive orderings, we first train a decoder with Common-First order on one batch of MS-COCO until it perfectly generates each sentence. We then fix the decoder and initialize an encoder. We train the encoder for 15k gradient steps using the procedure in Algorithm 1 to recover the ground truth Common-First order, and we report the final Normalized Levenshtein Distance against the ground truth in Table 2. We observe that modeling q_ϕ with Gumbel-Matching distribution significantly outperforms modeling with Plackett-Luce, despite the former requiring denominator approximation. We also observe that under Gumbel-Matching modeling distribution, the relative position encoding in Transformer-XL significantly outperforms other encoding schemes. Thus we combine these two techniques in our architecture design.

In addition, we analyze how the choice of K, the number of latents per training sample, affects model performance. We use the same setting as above and apply Transformer-XL relative position encoding, and we report the results in Table 3. We observe that the encoder more accurately fits to the ground truth order as K increases, until a value of around 10. Since a very large K can slow the model down while only bringing marginal improvement, a choice of K from 4 to 10 is sufficient.

Table 3: Normalized Levenshtein Distance between the encoder ordering and the ground truth with respect to the choice of K.

K	2	3	4	10	20
$\mathcal{D}_{NLD}$	0.31	0.28	0.25	0.21	0.21

8 Conclusion

We propose, to our best knowledge, the first unsupervised learner that learns high-quality autoregressive orders through fully-parallelizable end-to-end training without domain-specific tuning. We propose a procedure named *Variational Order Inference* that uses the Variational Lower Bound with the space of autoregressive orderings as latent. Building on techniques in combinatorical optimization, we develop a practical policy gradient algorithm to optimize the encoder of the variational objective, and we propose an encoder architecture that conditions on training examples to output autoregressive orders. Empirical results demonstrate that our model is capable of discovering autoregressive orders that are competitive with or even better than fixed and predefined orders. In addition, the global and local analysis of the orderings learned through *Variational Order Inference* suggest that they resemble a type of *best-first* generation order, characterized by prioritizing the generation of *descriptive* tokens and deprioritizing the generation of *modifier* tokens.

REFERENCES

Roee Aharoni and Yoav Goldberg. Towards string-to-tree neural machine translation. *arXiv preprint arXiv:1704.04743*, 2017.

David Alvarez-Melis and Tommi S. Jaakkola. Tree-structured decoding with doubly-recurrent neural networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=HkYhZDqxg.

Nima Anari and Alireza Rezaei. A tight analysis of bethe approximation for permanent. *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 1434–1445, 2019.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.0473.

Yu Bao, Hao Zhou, Jiangtao Feng, Mingxuan Wang, Shujian Huang, Jiajun Chen, and Lei LI. Non-autoregressive transformer by position learning, 2019.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

William Chan, Nikita Kitaev, Kelvin Guu, Mitchell Stern, and Jakob Uszkoreit. Kermit: Generative insertion-based modeling for sequences, 2019.

Eugene Charniak, Kevin Knight, and Kenji Yamada. Syntax-based language models for statistical machine translation. In *Proceedings of MT Summit IX*, pp. 40–46. Citeseer, 2003.

Xi Chen, Nikhil Mishra, Mostafa Rohaninejad, and Pieter Abbeel. Pixelsnail: An improved autoregressive generative model. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 863–871. PMLR, 2018. URL http://proceedings.mlr.press/v80/chen18h.html.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans (eds.), *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pp. 1724–1734. ACL, 2014. doi: 10.3115/v1/d14-1179. URL https://doi.org/10.3115/v1/d14-1179.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2978–2988, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1285. URL https://www.aclweb.org/anthology/P19-1285.

Michael Denkowski and Alon Lavie. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the EACL 2014 Workshop on Statistical Machine Translation*, 2014.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A Smith. Recurrent neural network grammars. *arXiv preprint arXiv:1602.07776*, 2016.

Dmitrii Emelianenko, Elena Voita, and Pavel Serdyukov. Sequence modeling with unconstrained generation order. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pp. 7698–7709, 2019. URL http://papers.nips.cc/paper/8986-sequence-modeling-with-unconstrained-generation-order.

N. Ford, Daniel Duckworth, Mohammad Norouzi, and G. Dahl. The importance of generation order in language modeling. *ArXiv*, abs/1808.07910, 2018a.

Nicolas Ford, Daniel Duckworth, Mohammad Norouzi, and George E Dahl. The importance of generation order in language modeling. *arXiv preprint arXiv:1808.07910*, 2018b.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. volume 37 of *Proceedings of Machine Learning Research*, pp. 881–889, Lille, France, 07–09 Jul 2015. PMLR. URL http://proceedings.mlr.press/v37/germain15.html.

David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. English gigaword, 2003.

Aditya Grover, E. Wang, Aaron Zweig, and S. Ermon. Stochastic optimization of sorting networks via continuous relaxations. *ArXiv*, abs/1903.08850, 2019.

Jetic Gū, Hassan S. Shavarani, and Anoop Sarkar. Top-down tree structured decoding with syntactic connections for neural machine translation and parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 401–413, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1037. URL https://www.aclweb.org/anthology/D18-1037.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. Non-autoregressive neural machine translation. In *5th International Conference on Learning Representations*, 2018.

Jiatao Gu, Qi Liu, and Kyunghyun Cho. Insertion-based decoding with automatically inferred generation order. *Transactions of the Association for Computational Linguistics*, 7:661–676, 2019a.

Jiatao Gu, Changhan Wang, and Junbo Zhao. Levenshtein transformer. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32, pp. 11181–11191. Curran Associates, Inc., 2019b. URL https://proceedings.neurips.cc/paper/2019/file/675f9820626f5bc0afb47b57890b466e-Paper.pdf.

Irina Higgins, Loïc Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Sy2fzU9gl.

Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3128–3137, 2015.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pp. 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W04-1013.

Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

Scott Linderman, Gonzalo Mena, Hal Cooper, Liam Paninski, and John Cunningham. Reparameterizing the birkhoff polytope for variational permutation inference. volume 84 of *Proceedings of Machine Learning Research*, pp. 1618–1627, Playa Blanca, Lanzarote, Canary Islands, 09–11 Apr 2018. PMLR. URL http://proceedings.mlr.press/v84/linderman18a.html.

W. Ling, P. Blunsom, Edward Grefenstette, K. Hermann, Tomás Kociský, Fumin Wang, and A. Senior. Latent predictor networks for code generation. *ArXiv*, abs/1603.06744, 2016.

R. Duncan Luce. *Individual Choice Behavior: A Theoretical analysis*. Wiley, New York, NY, USA, 1959.

Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In Lluís Màrquez, Chris Callison-Burch, Jian Su, Daniele Pighin, and Yuval Marton (eds.), *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pp. 1412–1421. The Association for Computational Linguistics, 2015. doi: 10.18653/v1/d15-1166. URL https://doi.org/10.18653/v1/d15-1166.

Xuezhe Ma, Chunting Zhou, Xian Li, Graham Neubig, and Eduard Hovy. FlowSeq: Non-autoregressive conditional sequence generation with generative flow. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4282–4292, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1437. URL https://www.aclweb.org/anthology/D19-1437.

Sébastien Marcel and Yann Rodriguez. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, pp. 1485–1488, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605589336. doi: 10.1145/1873951.1874254. URL https://doi.org/10.1145/1873951.1874254.

Shikib Mehri and Leonid Sigal. Middle-out decoding. In *Advances in Neural Information Processing Systems*, pp. 5518–5529, 2018.

Gonzalo Mena, David Belanger, Scott Linderman, and Jasper Snoek. Learning latent permutations with gumbel-sinkhorn networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=Byt3oJ-0W.

Gonzalo Mena, Erdem Varol, Amin Nejatbakhsh, Eviatar Yemini, and Liam Paninski. Sinkhorn permutation variational marginal inference. volume 118 of *Proceedings of Machine Learning Research*, pp. 1–9. PMLR, 08 Dec 2020. URL http://proceedings.mlr.press/v118/mena20a.html.

Tomáš Mikolov et al. Statistical language models based on neural networks. *Presentation at Google, Mountain View, 2nd April*, 80:26, 2012.

James R. Munkres. Algorithms for the Assignment and Transportation Problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, March 1957.

Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. Learning to generate pseudo-code from source code using statistical machine translation. In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ASE '15, pp. 574–584, Lincoln, Nebraska, USA, November 2015. IEEE Computer Society. ISBN 978-1-5090-0025-8. doi: 10.1109/ASE.2015.36. URL https://doi.org/10.1109/ASE.2015.36.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pp. 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL https://www.aclweb.org/anthology/P02-1040.

Robin L Plackett. The analysis of permutations. pp. 193–202, 1975.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training, 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

Laura Ruis, Mitchell Stern, Julia Proskurnia, and William Chan. Insertion-deletion transformer. *CoRR*, abs/2001.05540, 2020. URL https://arxiv.org/abs/2001.05540.

Alexander M. Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015. doi: 10.18653/v1/d15-1044. URL http://dx.doi.org/10.18653/v1/D15-1044.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL https://www.aclweb.org/anthology/P16-1162.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pp. 464–468, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-2074. URL https://www.aclweb.org/anthology/N18-2074.

Richard Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *Ann. Math. Statist.*, 35(2):876–879, 06 1964. doi: 10.1214/aoms/1177703591. URL https://doi.org/10.1214/aoms/1177703591.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. 15(1):1929–1958, January 2014. ISSN 1532-4435.

Felix Stahlberg. Neural machine translation: A review. *ArXiv*, abs/1912.02047, 2019.

Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. Insertion transformer: Flexible sequence generation via insertion operations. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 5976–5985. PMLR, 2019. URL http://proceedings.mlr.press/v97/stern19a.html.

Qing Sun, Stefan Lee, and Dhruv Batra. Bidirectional beam search: Forward-backward inference in neural sequence models for fill-in-the-blank image captioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6961–6969, 2017.

Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *ICML*, 2011.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 3104–3112, 2014a. URL http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014b.

Richard S Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. A. Solla, T. K. Leen, and K. Müller (eds.), *Advances in Neural Information Processing Systems 12*, pp. 1057–1063. MIT Press, 2000. URL http://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximatio pdf.

Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation. *J. Mach. Learn. Res.*, 17:205:1–205:37, 2016. URL http://jmlr.org/papers/v17/16-272.html.

Aäron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Koray Kavukcuoglu, Oriol Vinyals, and Alex Graves. Conditional image generation with pixelcnn decoders. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 4790–4798, 2016. URL http://papers.nips.cc/paper/6527-conditional-image-generation-with-pixelcnn-decoders.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 5998–6008. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

Ramakrishna Vedantam, C. Lawrence Zitnick, and Devi Parikh. Cider: Consensus-based image description evaluation. In *CVPR*, pp. 4566–4575. IEEE Computer Society, 2015. ISBN 978-1-4673-6964-0. URL http://dblp.uni-trier.de/db/conf/cvpr/cvpr2015.html#VedantamZP15.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 2692–2700, 2015a. URL http://papers.nips.cc/paper/5866-pointer-networks.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pp. 3156–3164. IEEE Computer Society, 2015b. doi: 10.1109/CVPR.2015.7298935. URL https://doi.org/10.1109/CVPR.2015.7298935.

Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. In Yoshua Bengio and Yann LeCun (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.06391.

P. O. Vontobel. The bethe permanent of a non-negative matrix. In *2010 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 341–346, 2010.

Xinyi Wang, Hieu Pham, Pengcheng Yin, and Graham Neubig. A tree-based decoder for neural machine translation. *arXiv preprint arXiv:1808.09374*, 2018.

Sean Welleck, Kianté Brantley, Hal Daumé III, and Kyunghyun Cho. Non-monotonic sequential text generation. *arXiv preprint arXiv:1902.02192*, 2019a.

Sean Welleck, Kianté Brantley, Hal Daumé III, and Kyunghyun Cho. Non-monotonic sequential text generation. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6716–6726. PMLR, 2019b. URL http://proceedings.mlr.press/v97/welleck19a.html.

Lijun Wu, Xu Tan, Di He, Fei Tian, Tao Qin, Jianhuang Lai, and Tie-Yan Liu. Beyond error propagation in neural machine translation: Characteristics of language also matter. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3602–3611, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1396. URL https://www.aclweb.org/anthology/D18-1396.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In Francis R. Bach and David M. Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pp. 2048–2057. JMLR.org, 2015. URL http://proceedings.mlr.press/v37/xuc15.html.

Kenji Yamada and Kevin Knight. A syntax-based statistical translation model. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pp. 523–530, 2001.

Long Zhou, Jiajun Zhang, and Chengqing Zong. Synchronous bidirectional neural machine translation. *Transactions of the Association for Computational Linguistics*, 7:91–105, 2019.

APPENDIX

A GUMBEL-MATCHING DISTRIBUTION AND ITS SAMPLING

In Section 4, We model the distribution of $q_\phi(\cdot|\mathbf{y}, \mathbf{x})$ as a Gumbel-Matching distribution $\mathcal{G}.\mathcal{M}.(X)$ over $\mathcal{P}_{n\times n}$, where $X = \phi(\mathbf{y}, \mathbf{x}) \in \mathbb{R}^{n\times n}$ is the latent output.

To obtain samples in $\mathcal{P}_{n\times n}$ from the Gumbel-Matching distribution, Mena et al. (2018) relaxes $\mathcal{P}_{n\times n}$ to $\mathcal{B}_{n\times n}$ by defining the Gumbel-Sinkhorn distribution $\mathcal{G}.\mathcal{S}.(X, \tau) : \tau > 0$ over $\mathcal{B}_{n\times n}$. Here we reproduce the following definitions and theorems with similar notations from Sinkhorn (1964) and Mena et al. (2018):

Definition A.1. *Let $X \in \mathbb{R}_{n\times n}$ and $A \in \mathbb{R}^+_{n\times n}$. The **Sinkhorn Operator** S is defined as*

$$\mathcal{T}_r(A) = A \oslash (A\mathbf{1}_n\mathbf{1}_n^T) \tag{8}$$

$$\mathcal{T}_c(A) = A \oslash (\mathbf{1}_n\mathbf{1}_n^T A) \tag{9}$$

$$\mathcal{T}(A) = \mathcal{T}_r(\mathcal{T}_c(A)) \tag{10}$$

$$S(X) = \lim_{n\to\infty} \mathcal{T}^n(\exp(X)) \tag{11}$$

Here, $\oslash$ is the element-wise division between two matrices, and $\mathcal{T}_r$ and $\mathcal{T}_c$ are row and column normalizations of a non-negative matrix, respectively. Therefore, iteratively applying $\mathcal{T}$ is equivalent to iteratively normalizing a non-negative matrix by column and row.

Theorem A.2. (Sinkhorn, 1964) *The range of S is $\mathcal{B}_{n\times n}$.*

Theorem A.3. (Mena et al., 2018) *Let $X \in \mathbb{R}_{n\times n}, \tau > 0$. The **Gumbel-Sinkhorn distribution** $\mathcal{G}.\mathcal{S}.(X, \tau)$ is defined as follows:*

$$\mathcal{G}.\mathcal{S}.(X, \tau) = S(\frac{X + \epsilon}{\tau}) \tag{12}$$

where ϵ is a matrix of i.i.d. standard Gumbel noise. Moreover, $\mathcal{G}.\mathcal{S}.(X, \tau)$ converges almost surely to $\mathcal{G}.\mathcal{M}.(X)$ as $\tau \to 0^+$.

To approximately sample from $\mathcal{G}.\mathcal{M}.(X)$, we first sample from $\mathcal{G}.\mathcal{S}.(X, \tau)$. Even though theoretically, $\mathcal{T}$ needs to be applied infinite number of times to obtain a matrix in $\mathcal{B}_{n\times n}$, Mena et al. (2018) reports that 20 iterations of $\mathcal{T}$ are enough in practice. We find that in our experiments, 20 iterations are not enough to obtain a matrix in $\mathcal{B}_{n\times n}$, but $100 - 200$ iterations are enough. After we obtain the matrix in $\mathcal{B}_{n\times n}$, we apply Hungarian algorithm (Munkres, 1957) to obtain $P \in \mathcal{G}.\mathcal{M}.(X)$.

Finally, we need to calculate the entropy term $\mathcal{H}_{q_\phi}$ in L_ϕ in Equation 3. This can be approximated using the technique in Appendix B.3 of Mena et al. (2018).

B MATRIX PERMANENT AND ITS APPROXIMATION WITH BETHE PERMANENT

In this section, we present details about matrix permanent and bethe permanent, which we use as an approximation to the denominator of $q_\phi(\cdot|\mathbf{y}, \mathbf{x})$.

Definition B.1. *Let $A \in \mathbb{R}_{n\times n}$. The **permanent** of A is defined as follows:*

$$perm(A) = \sum_{\sigma\in S_n} \prod_{i=1}^{n} A_{i,\sigma_i} \tag{13}$$

Theorem B.2. *The denominator of $q_\phi(\cdot|\mathbf{y}, \mathbf{x})$ equals $perm(\exp(X))$.*

Proof.

$$\begin{aligned}\sum_{P\in\mathcal{P}_{n\times n}} \exp\langle X, P\rangle_F &= \sum_{\sigma\in S_n} \exp(\sum_{i=1}^{n} X_{i,\sigma(i)}) \\ &= \sum_{\sigma\in S_n}\prod_{i=1}^{n}(\exp(X))_{i,\sigma(i)} \\ &= \text{perm}(\exp(X))\end{aligned}$$

□

Definition B.3. (Vontobel, 2010; Anari & Rezaei, 2019) *Let $A \in \mathbb{R}^{+}_{n\times n}$. The **bethe permanent** of A is defined as follows:*

$$perm_B(A) = \exp\left(\max_{\gamma\in\mathcal{B}_{n\times n}} \sum_{i,j}(\gamma_{i,j}\log A_{i,j} - \gamma_{i,j}\log\gamma_{i,j} + (1-\gamma_{i,j})\log(1-\gamma_{i,j}))\right) \tag{14}$$

Theorem B.4. (Anari & Rezaei, 2019) *Let $A \in \mathbb{R}^{+}_{n\times n}$. Then, $\sqrt{2}^{-n} perm(A) \leq perm_B(A) \leq perm(A)$.*

The γ in Definition B.3 can be calculated using the message passing algorithm in Lemma 29 of Vontobel (2010). An efficient implementation has recently been introduced in Appendix C of Mena et al. (2020). Therefore, we can use $\text{perm}_B(\exp(X))$ to approximate the denominator of $q_\phi(\cdot|\mathbf{y},\mathbf{x})$, and we can then use policy gradient to compute $\nabla_\phi L(\phi)$ in Equation (3).

C Hyperparameter Settings

We train on Django for 80k gradient steps, MS-COCO for 300k gradient steps, and Gigaword for 1M gradient steps. We apply Adam Optimizer (Kingma & Ba, 2015) with $\beta_1 = 0.99, \beta_2 = 0.999$ for MS-COCO, and $\beta_1 = 0.99, \beta_2 = 0.98$ for all other tasks. For baseline experiments, we use a batch size of 64 for Django and MS-COCO, and 128 for Gigaword. We decrease the learning rate linearly from 1e-4 to zero.

For our model, we use a batch size of 32 for MS-COCO and Django and 50 for Gigaword. We train the encoder and the decoder with shared embedding for the first 25% of steps and separated embedding for rest of the steps. We set the decoder learning rate to be 5e-5 and encoder learning rate to be 5e-6. When the embedding is shared, we set the entropy coefficient $\beta = 0.3$ for all tasks. After we separate the embedding, for MS-COCO, we anneal β with a log-linear schedule from 0.3 to 0.03. For Django, we adopt the same schedule for the first 90% of steps and then decreases β to 0.003 for the rest of the steps. We find that this schedule allows the encoder to commit to a single ordering on sequences of longer length and slightly improves performance. For Gigaword, we add a cosine alignment loss between the decoder and the encoder vocabulary. Since the Gigaword vocabulary is much larger than that of MS-COCO and Django and contains many rare words, this loss allows the encoder to leverage the semantic information recently-learnt from the decoder to better discover autoregressive orderings. We anneal β log-linearly from 0.3 to 0.03 in 400k gradient steps. We then fix the encoder and fine-tune the decoder with a batch size of 128 with learning rate linearly decreasing from 7e-5 to 0.

During experiments, we observe that the KL-divergence between the encoder latent distribution and the uniform distribution first decreases till near-zero and then increases as the encoder converges to smaller set of orderings for each training sample. When at least 10% of the sampled permutations in a single batch are not unique, we decrease the decoder learning rate to 3e-5 and encoder learning rate to 3e-6.

In total, it takes about 50k gradient steps for the encoder to converge on Django, about 200k gradient steps on MS-COCO, and about 500k gradient steps on Gigaword.

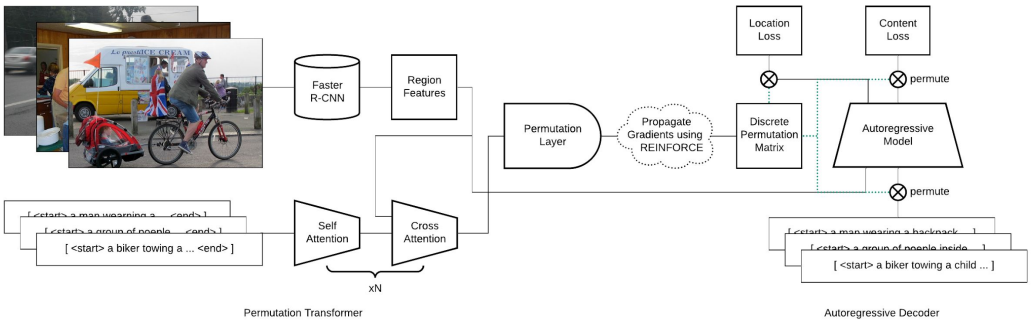


Figure 5: This figure demonstrates our algorithm for an image captioning task. The model on the left is the Permutation Transformer, which maps training examples to permutation matrices. The model on the right is an autoregressive model that learns to predict tokens and positions.

D Example Architecture for Conditional Sequence Generation

In Section 5, we introduced the specific encoder and decoder architectures used for the conditional sequence generation tasks in our paper. To further illustrate the architecture of *Variational Order Inference* , we present a diagram of the architecture instantiated for COCO 2017.

E Parts Of Speech Mappings

The parts of speech used in our Order Analysis section correspond to the NLTK Universal Tagset. In the below table, we provide mappings for the tag identifiers used in our main paper. More information about the specific NLTK tags can be found at the following url: http://www.nltk.org/book/ch05.html.

Tag	Meaning	English Examples
ADJ	adjective	new, good, high, special, big, local
ADP	adposition	on, of, at, with, by, into, under
ADV	adverb	really, already, still, early, now
CONJ	conjunction	and, or, but, if, while, although
DET	determiner, article	the, a, some, most, every, no, which
NOUN	noun	year, home, costs, time, Africa
NUM	numeral	twenty-four, fourth, 1991, 14:24
PRT	particle	at, on, out, over per, that, up, with
PRON	pronoun	he, their, her, its, my, I, us
VERB	verb	is, say, told, given, playing, would
.	punctuation marks	. , ; !
X	other	ersatz, esprit, dunno, gr8, univeristy

Table 4: NLTK Universal Tagset.

F Visualizations of Sequence Generation

F.1 COCO

We visualize the generation order inferred by *Variational Order Inference* for COCO. Sequences are generated using beam search over both tokens and their insertion positions, using a beam size of 3. Bounding boxes that correspond to region-features calculated using bottom-up attention are superimposed on the image, with an opacity value proportional to the magnitude of their softmax attention value in the final cross-attention layer in the language model.

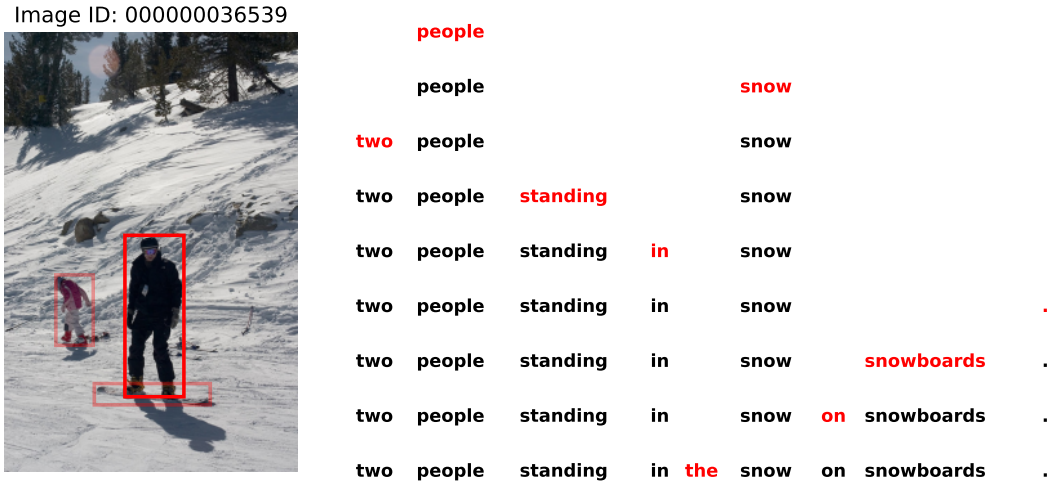


Figure 6: Generation order inferred by **Variational Order Inference**. Without supervision over its generation order, nor a domain-specific initialization, nor a prior to aid learning, the model learns an adaptive strategy that prioritizes object names—in this case, *people* and *snow*.

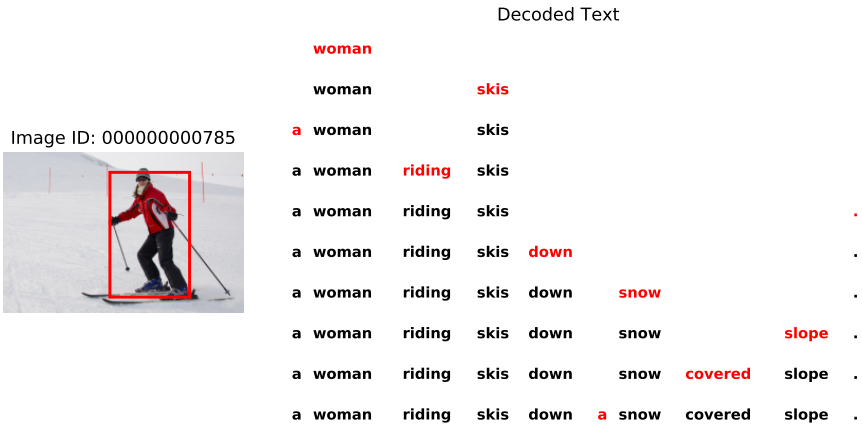


Figure 7: Generation order inferred by **Ours-VOI** for an image from the COCO 2017 validation set with the image identifier **000000000785**.

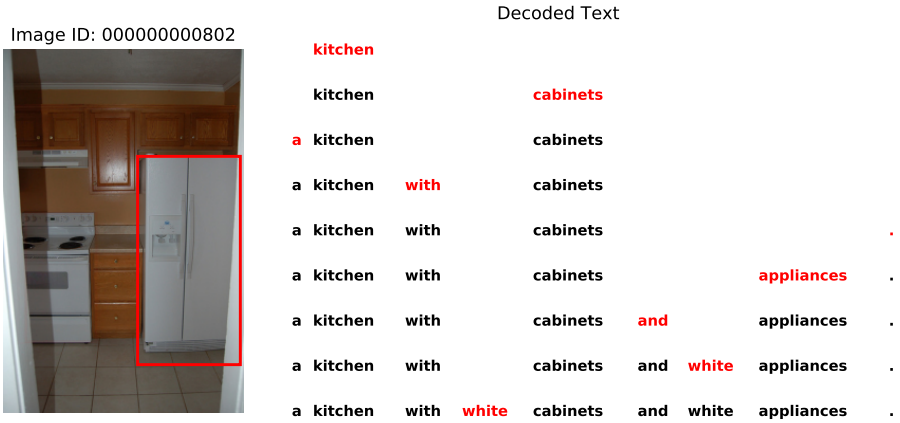


Figure 8: Generation order inferred by **Ours-VOI** for an image from the COCO 2017 validation set with the image identifier **000000000802**.

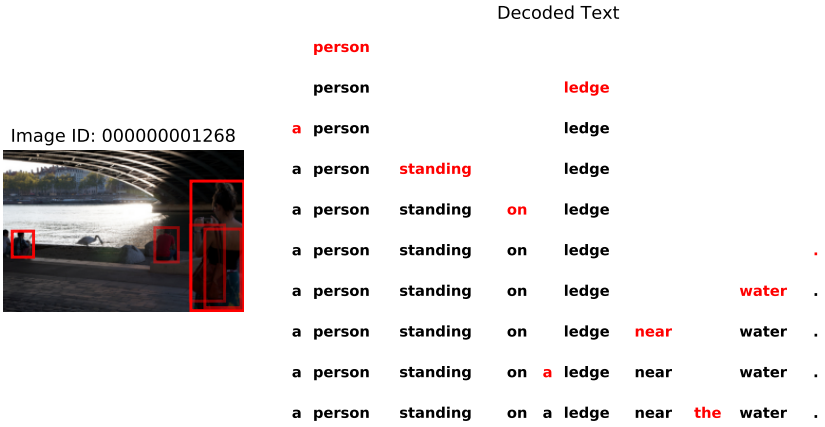


Figure 9: Generation order inferred by **Ours-VOI** for an image from the COCO 2017 validation set with the image identifier **000000001268**.

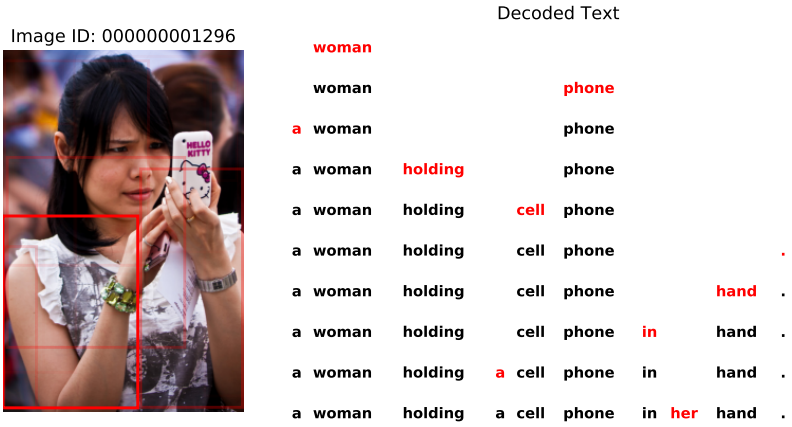


Figure 10: Generation order inferred by **Ours-VOI** for an image from the COCO 2017 validation set with the image identifier **000000001296**.

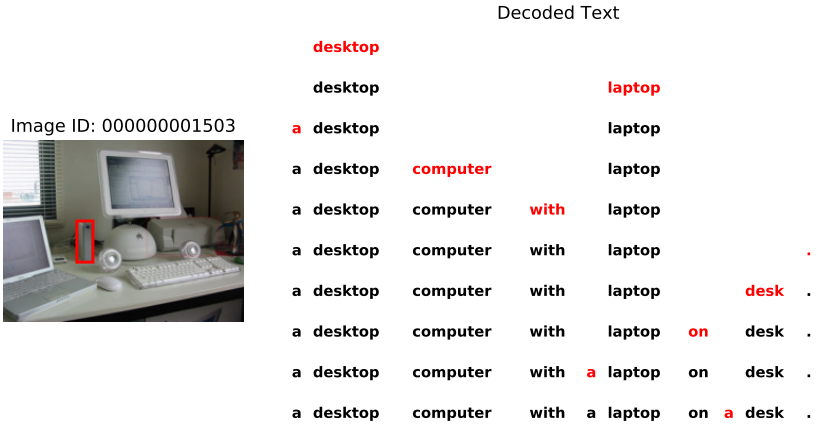


Figure 11: Generation order inferred by **Ours-VOI** for an image from the COCO 2017 validation set with the image identifier **000000001503**.

Figure 12: Generation order inferred by **Ours-VOI** for an image from the COCO 2017 validation set with the image identifier **000000001993**.

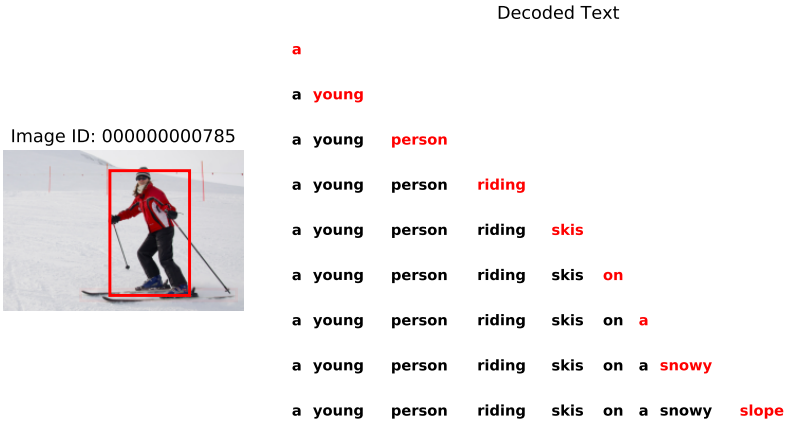


Figure 13: Generation order inferred by **Ours-L2R** for an image from the COCO 2017 validation set with the image identifier **000000000785**.

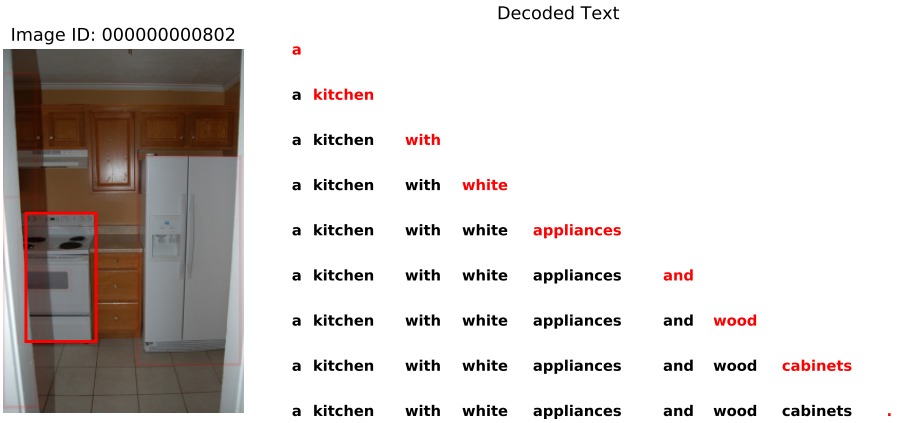


Figure 14: Generation order inferred by **Ours-L2R** for an image from the COCO 2017 validation set with the image identifier **000000000802**.

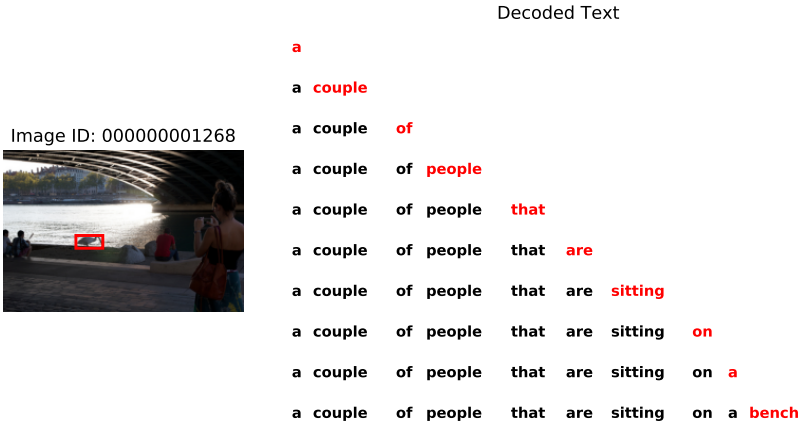


Figure 15: Generation order inferred by **Ours-L2R** for an image from the COCO 2017 validation set with the image identifier **000000001268**.

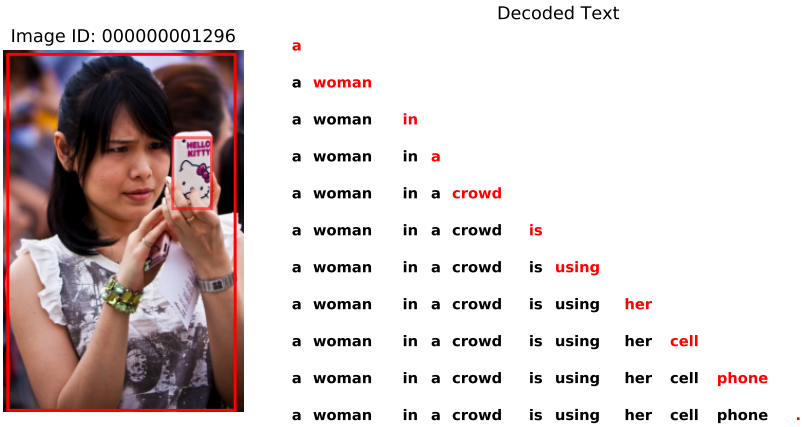


Figure 16: Generation order inferred by **Ours-L2R** for an image from the COCO 2017 validation set with the image identifier **000000001296**.

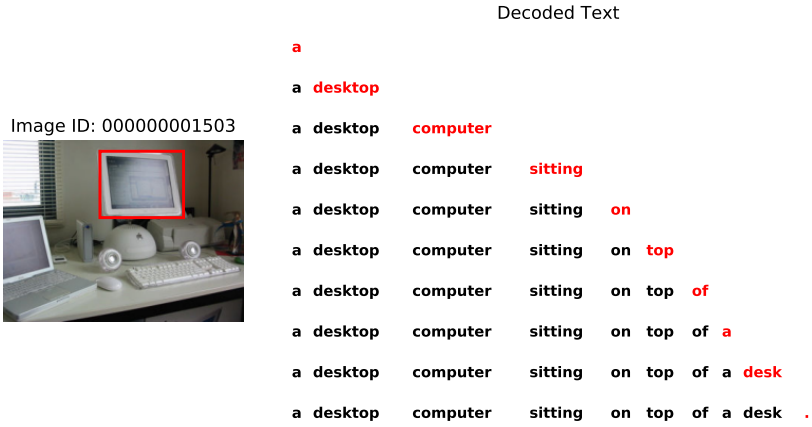


Figure 17: Generation order inferred by **Ours-L2R** for an image from the COCO 2017 validation set with the image identifier **000000001503**.

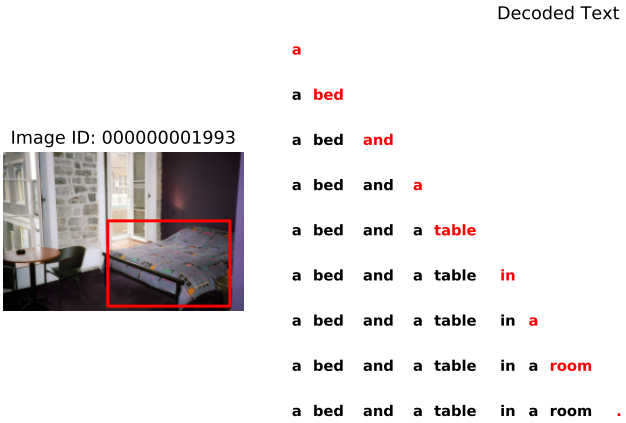


Figure 18: Generation order inferred by **Ours-L2R** for an image from the COCO 2017 validation set with the image identifier **000000001993**.

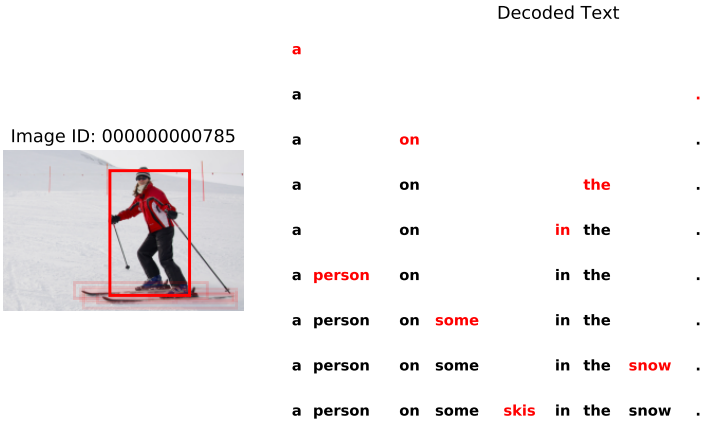


Figure 19: Generation order inferred by **Ours-Common** for an image from the COCO 2017 validation set with the image identifier **000000000785**.

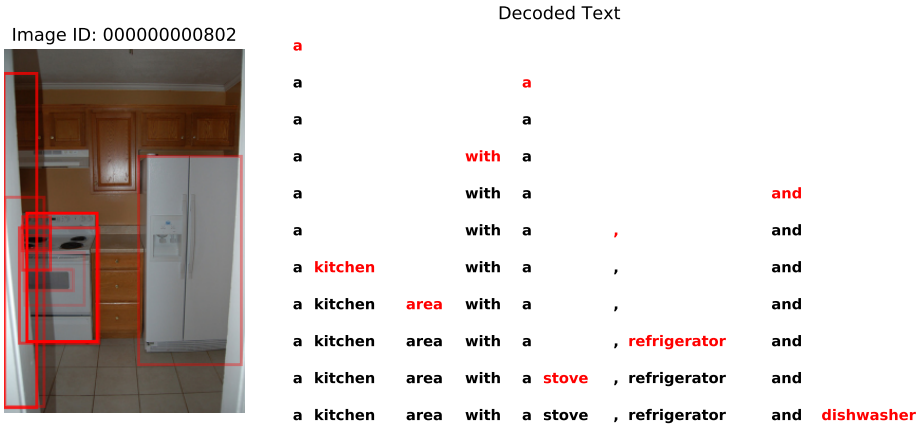


Figure 20: Generation order inferred by **Ours-Common** for an image from the COCO 2017 validation set with the image identifier **000000000802**.

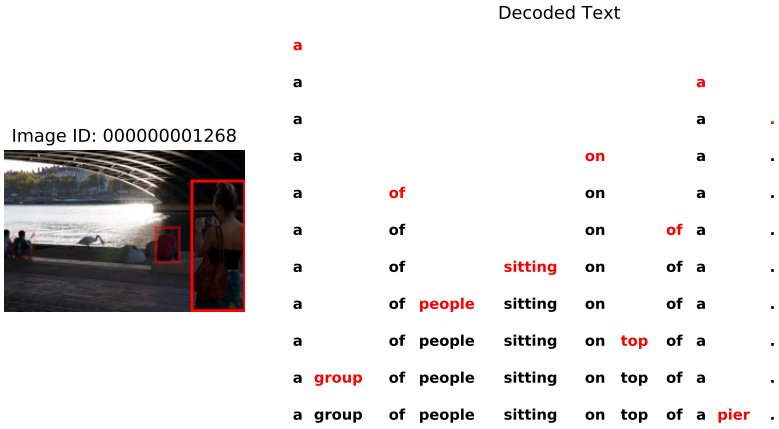


Figure 21: Generation order inferred by **Ours-Common** for an image from the COCO 2017 validation set with the image identifier **000000001268**.

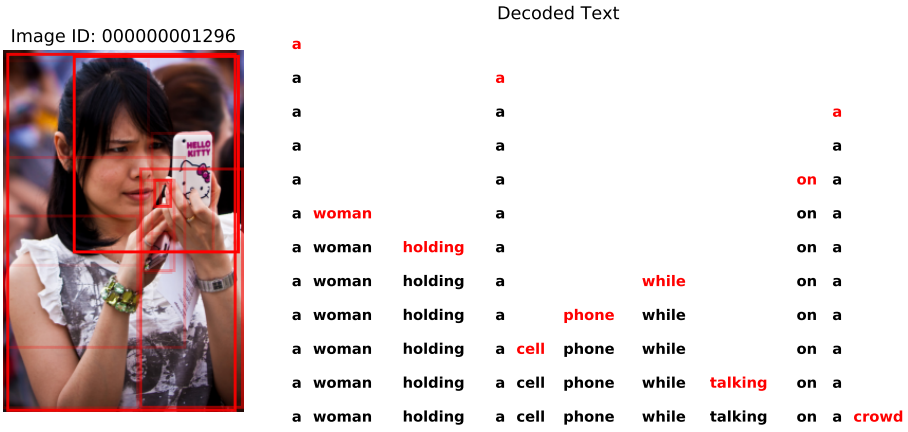


Figure 22: Generation order inferred by **Ours-Common** for an image from the COCO 2017 validation set with the image identifier **000000001296**.

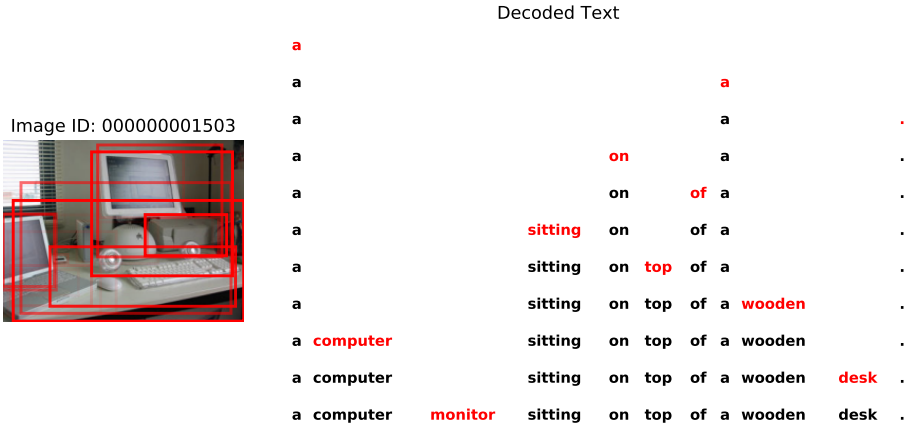


Figure 23: Generation order inferred by **Ours-Common** for an image from the COCO 2017 validation set with the image identifier **000000001503**.

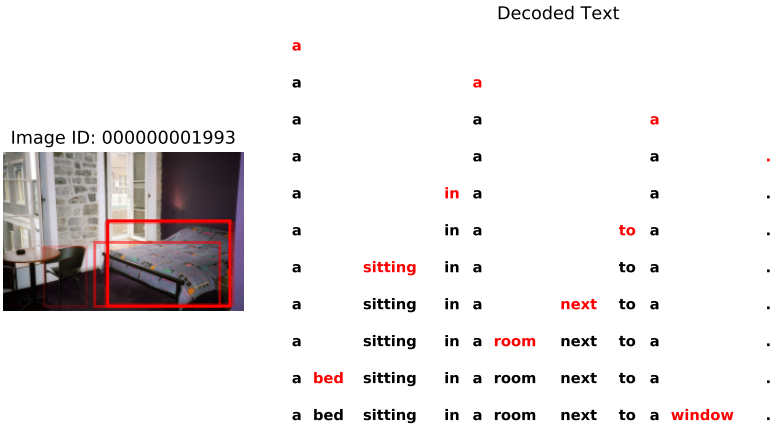


Figure 24: Generation order inferred by **Ours-Common** for an image from the COCO 2017 validation set with the image identifier **000000001993**.

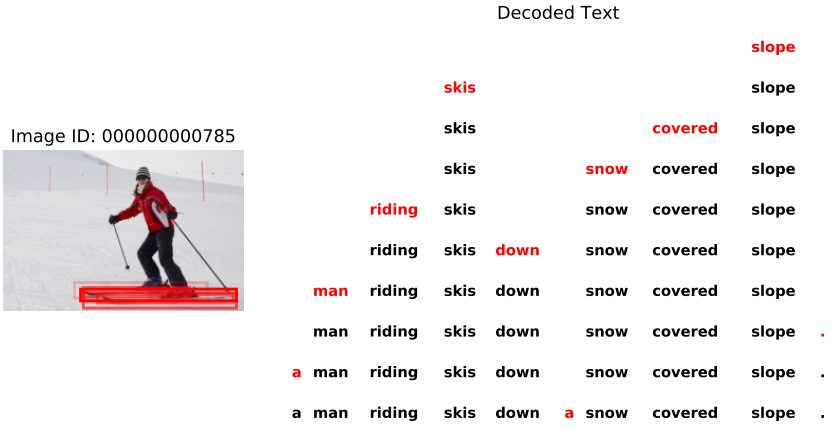


Figure 25: Generation order inferred by **Ours-Rare** for an image from the COCO 2017 validation set with the image identifier **000000000785**.

Figure 26: Generation order inferred by **Ours-Rare** for an image from the COCO 2017 validation set with the image identifier **000000000802**.

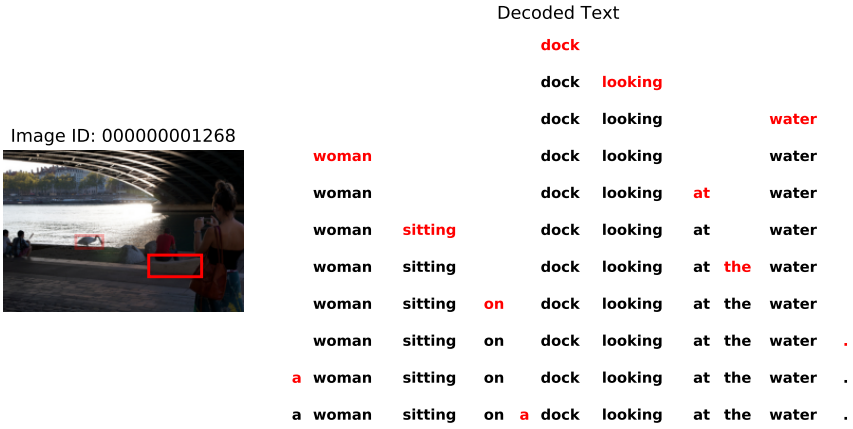


Figure 27: Generation order inferred by **Ours-Rare** for an image from the COCO 2017 validation set with the image identifier **000000001268**.

Figure 28: Generation order inferred by **Ours-Rare** for an image from the COCO 2017 validation set with the image identifier **000000001296**.

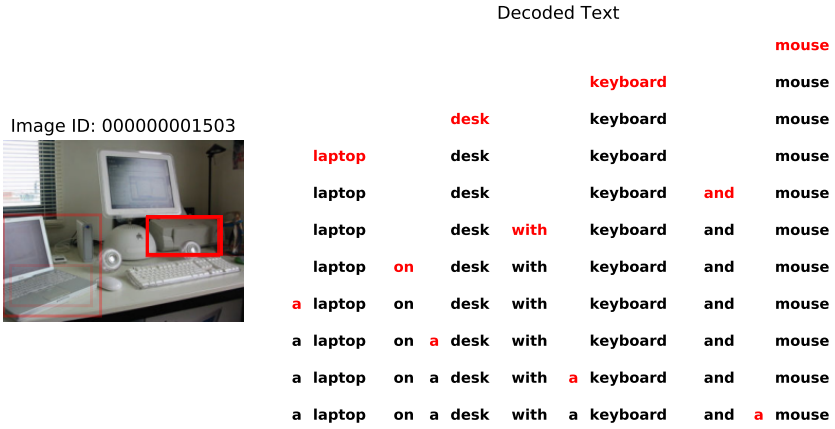
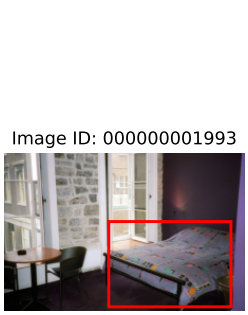


Figure 29: Generation order inferred by **Ours-Rare** for an image from the COCO 2017 validation set with the image identifier **000000001503**.



Decoded Text

							tables	
					chairs		tables	
				bed	chairs		tables	
	room			bed	chairs		tables	
	room			bed	, chairs		tables	
	room			bed	, chairs	,	tables	
	room			bed	, chairs	, and	tables	
	room	with		bed	, chairs	, and	tables	
	room	with		bed	, chairs	, and	tables	.
a	room	with		bed	, chairs	, and	tables	.
a	room	with	a	bed	, chairs	, and	tables	.

Figure 30: Generation order inferred by **Ours-Rare** for an image from the COCO 2017 validation set with the image identifier **000000001993**.

F.2 DJANGO

We visualize the latent generation order inferred by Variational Order Inference for Django. Sequences are generated using a beam search over both the tokens and their insertion positions, using a beam size of 3. Text on which the model is conditioned is provided on the left for each example.

Conditioned Text

raise an AttributeError with an argument string _STR:0_ , formated with self.name [self . name] .

Decoded Text

	AttributeError									
	AttributeError								name	
	AttributeError		(						name	
	AttributeError		(	'_STR:0_'					name	
	AttributeError		(	'_STR:0_'	%				name	
	AttributeError		(	'_STR:0_'	%	self			name	
	AttributeError		(	'_STR:0_'	%	self	.		name	
	AttributeError		(	'_STR:0_'	%	self	.		name	)
raise	AttributeError		(	'_STR:0_'	%	self	.		name	)

Figure 31: Generation order inferred by **Ours-VOI** for a pseudocode sample from the Django natural language to code test set with the sample id **154**.

F.3 GIGAWORD

We visualize the latent generation order inferred by Variational Order Inference for Gigaword. Sequences are generated using a beam search over both the tokens and their insertion positions, using a beam size of 3. Text on which the model is conditioned is provided on the left for each example.

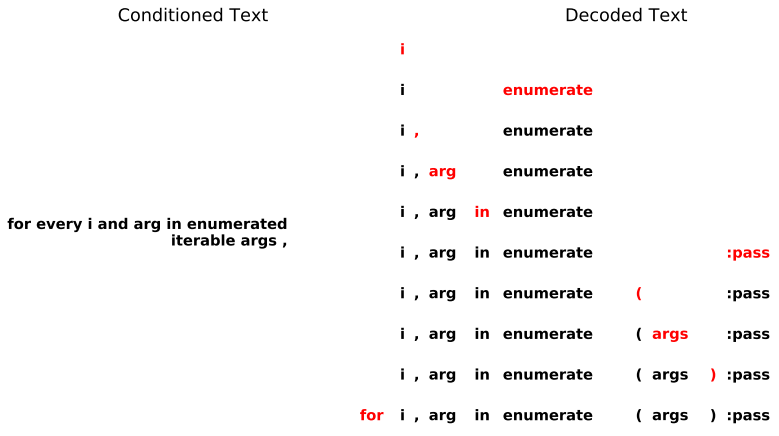


Figure 32: Generation order inferred by **Ours-VOI** for a pseudocode sample from the Django natural language to code test set with the sample id **431**.

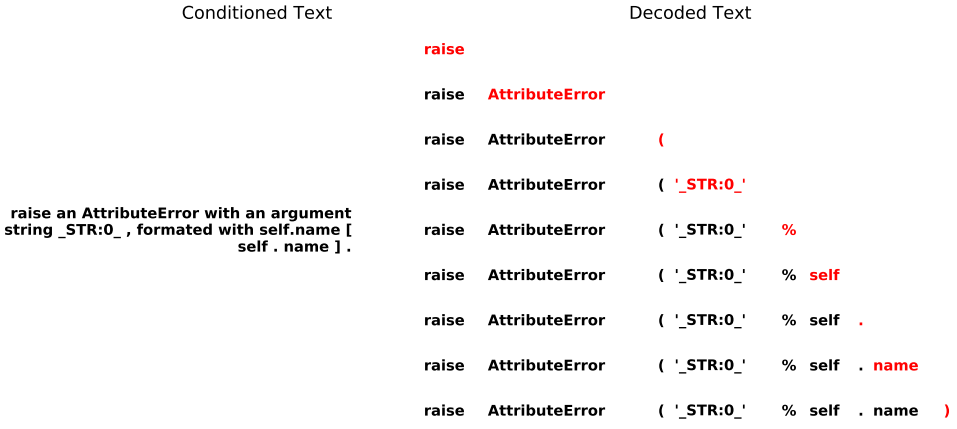


Figure 33: Generation order inferred by **Ours-L2R** for a pseudocode sample from the Django natural language to code test set with the sample id **154**.

Conditioned Text	Decoded Text							
for every i and arg in enumerated iterable args ,	for							
	for	i						
	for	i	,					
	for	i	,	arg				
	for	i	,	arg	in			
	for	i	,	arg	in	enumerate		
	for	i	,	arg	in	enumerate	(	
	for	i	,	arg	in	enumerate	(	args
	for	i	,	arg	in	enumerate	(	args)
	for	i	,	arg	in	enumerate	(	args) :pass

Figure 34: Generation order inferred by **Ours-L2R** for a pseudocode sample from the Django natural language to code test set with the sample id **431**.

Conditioned Text	Decoded Text										
raise an AttributeError with an argument string _STR:0_ , formated with self.name [self . name] .										.	
										.	.
									self	.	.
						(			self	.	.
						(			self	.	.
						(	'_STR:0_'		self	.	.
						(	'_STR:0_'	name	self	.	.
						(	'_STR:0_'	% name	self	.	.
	raise					(	'_STR:0_'	% name	self	.	.
	raise					(	'_STR:0_'	% name	self	.	.
	raise					(	'_STR:0_'	% name	self	. __class__	.
	raise	AttributeError				(	'_STR:0_'	% name	self	. __class__	.

Figure 35: Generation order inferred by **Ours-Common** for a pseudocode sample from the Django natural language to code test set with the sample id **154**.

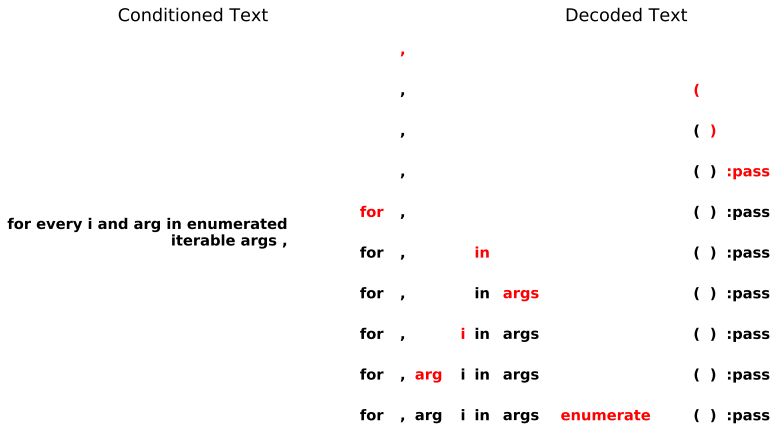


Figure 36: Generation order inferred by **Ours-Common** for a pseudocode sample from the Django natural language to code test set with the sample id **431**.

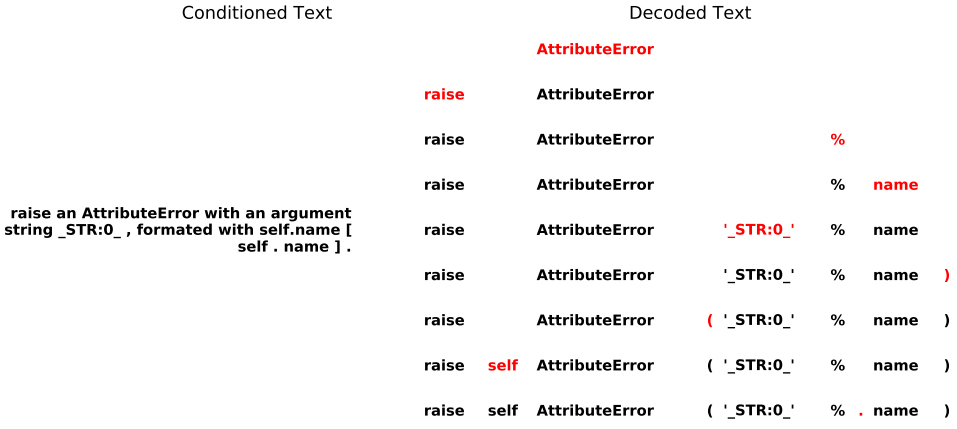


Figure 37: Generation order inferred by **Ours-Rare** for a pseudocode sample from the Django natural language to code test set with the sample id **154**.

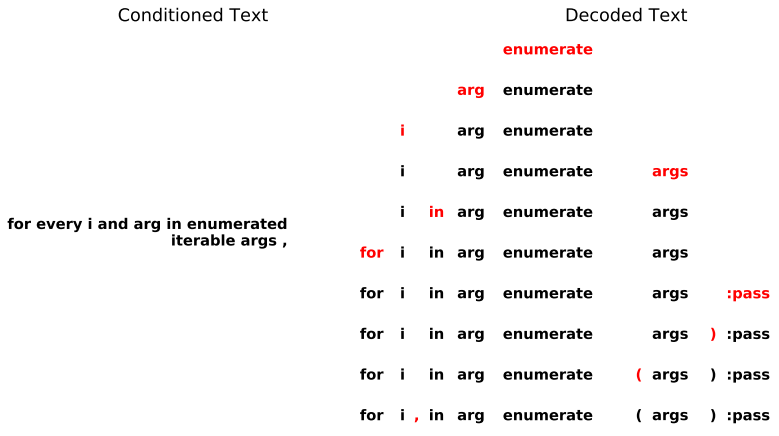


Figure 38: Generation order inferred by **Ours-Rare** for a pseudocode sample from the Django natural language to code test set with the sample id **431**.

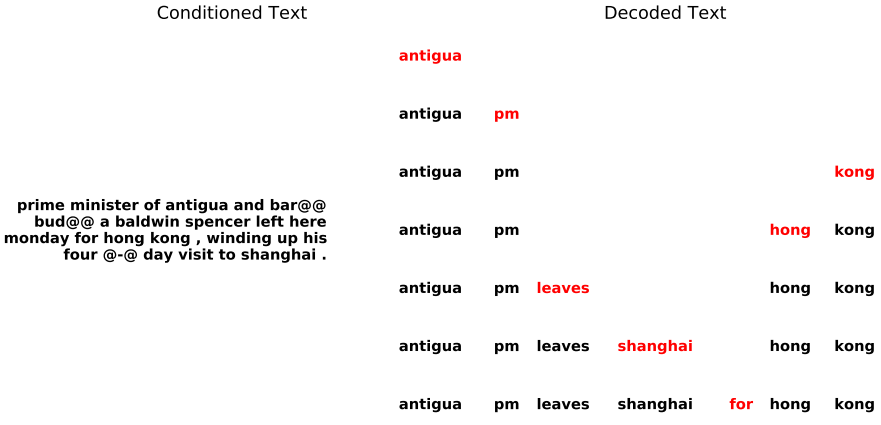


Figure 39: Generation order inferred by **Ours-VOI** for a text sample from the Gigaword text summarization test set with the sample id **15**.

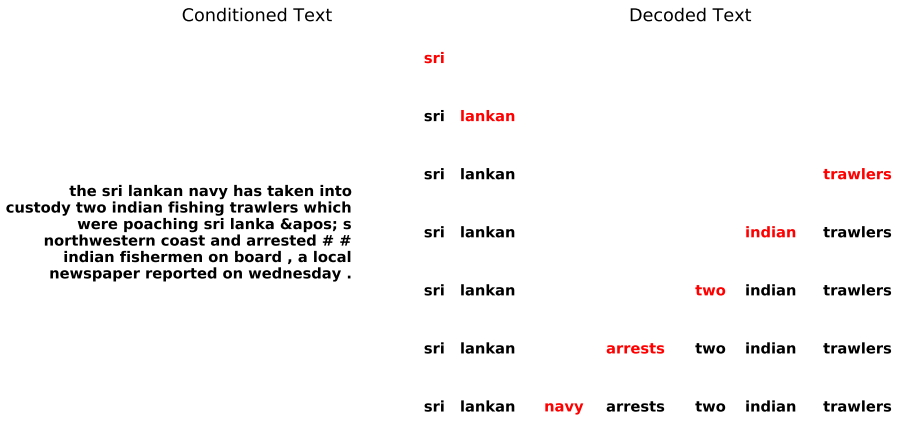


Figure 40: Generation order inferred by **Ours-VOI** for a text sample from the Gigaword text summarization test set with the sample id **33**.

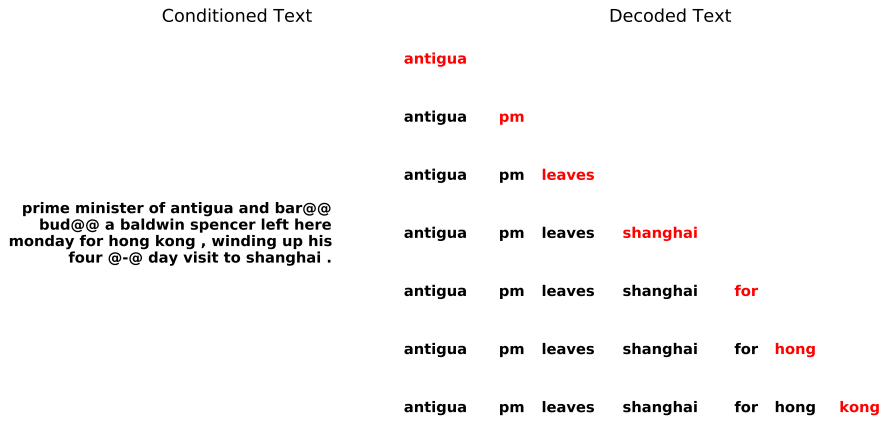


Figure 41: Generation order inferred by **Ours-L2R** for a text sample from the Gigaword text summarization test set with the sample id **15**.

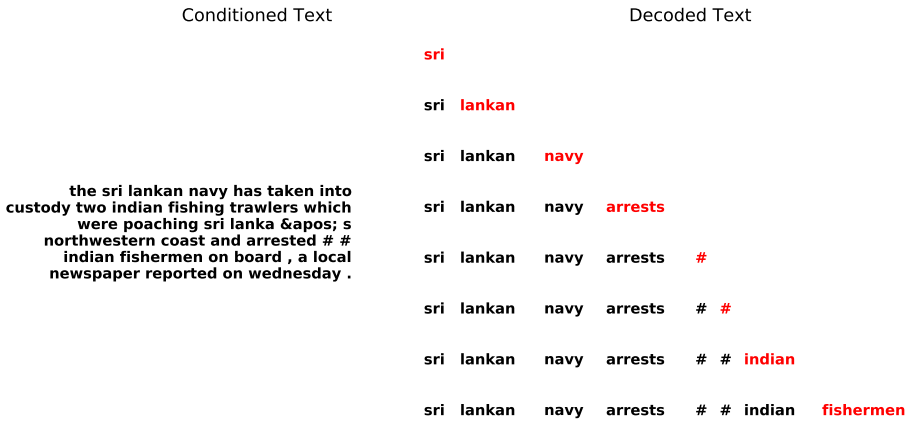


Figure 42: Generation order inferred by **Ours-L2R** for a text sample from the Gigaword text summarization test set with the sample id **33**.

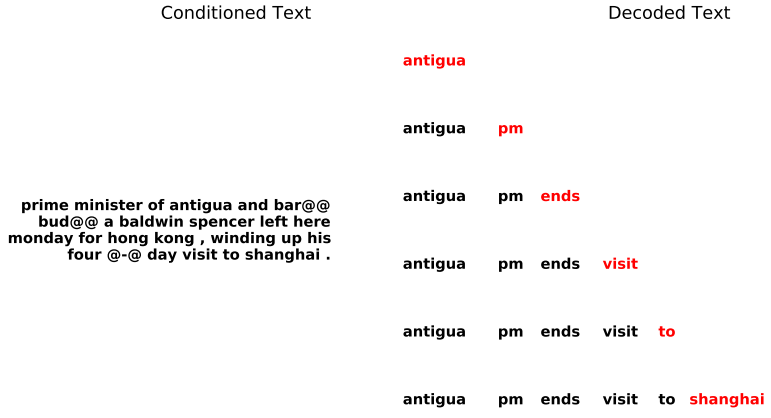


Figure 43: Generation order inferred by **Ours-Common** for a text sample from the Gigaword text summarization test set with the sample id **15**.

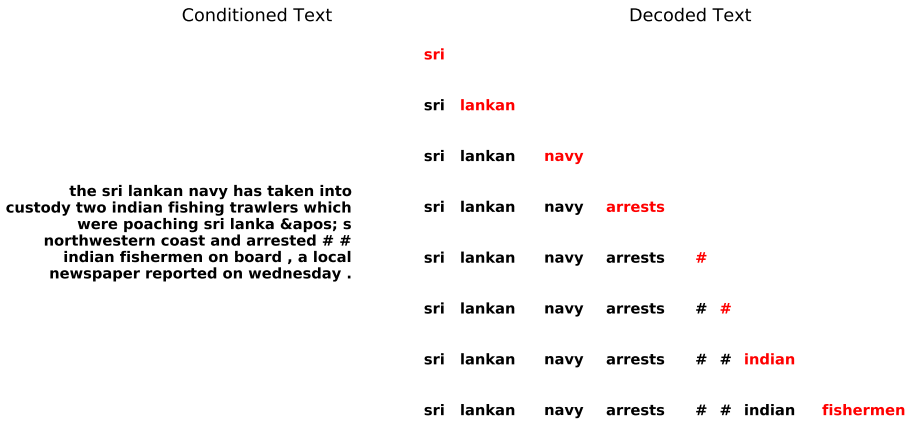


Figure 44: Generation order inferred by **Ours-Common** for a text sample from the Gigaword text summarization test set with the sample id **33**.

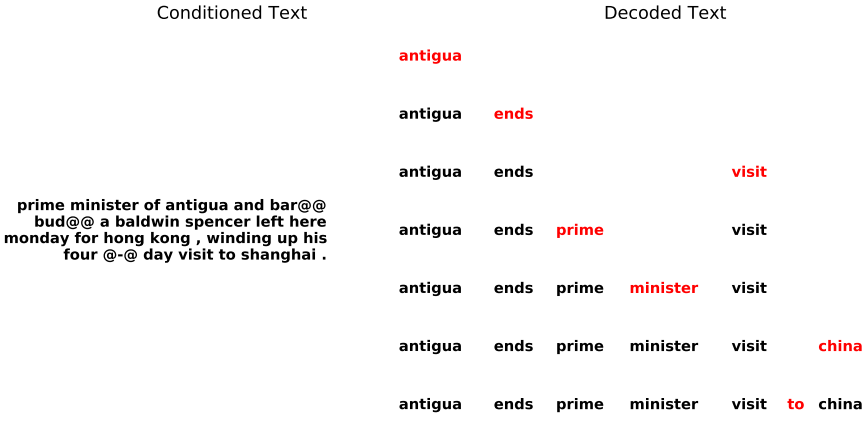


Figure 45: Generation order inferred by **Ours-Rare** for a text sample from the Gigaword text summarization test set with the sample id **15**.

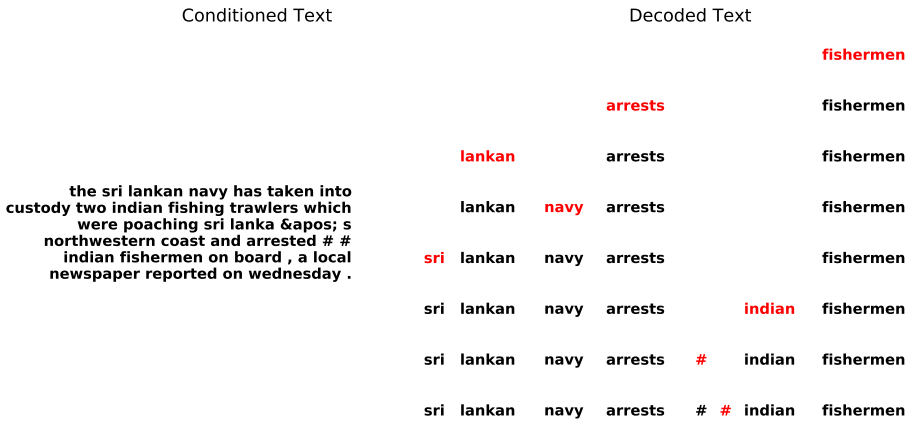


Figure 46: Generation order inferred by **Ours-Rare** for a text sample from the Gigaword text summarization test set with the sample id **33**.