Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Compositional Reasoning with Diffusion Models for Out-of-Distribution Generalization

Semester Thesis (Research in Data Science 6KP)

Michal Tešnar

mtesnar@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Joël Mathys
Prof. Dr. Roger Wattenhofer

April 5, 2026

Acknowledgements

This research was enabled in part by compute resources provided by Computer Engineering and Networks Laboratory (TIK) at ETH.

Abstract

To develop machine learning models capable of uncovering new knowledge, algorithms that surpass simple memorization and instead abstract and execute the underlying logic of a task need to be designed. This capability is best evaluated on out-of-distribution (OOD) instances, which measure whether model has learned more than a compression of the training data, but rather abstracted the principles leading to the solution. In this work, we focus on the Tents Puzzle, a variation of the constraint satisfaction problem, which can be presented in different sizes. We analyze the puzzle to design a data representation specifically to facilitate OOD generalization. We demonstrate that a standard transformer encoder fails to extrapolate beyond its training domain. Furthermore, its generalization performance is linked with increasing difficulty of the puzzle instances. Subsequently, we show that diffusion models achieve better results out of distribution by decoding the puzzle iteratively; however, these gains remain modest for test instances further from the training distribution. To address this, we decompose OOD samples into sub-problems that fall within the training distribution. By aggregating the solutions of these sub-problems, we achieve superior performance across all grid sizes, even significantly beyond the training domain. Our findings suggest that improvements in OOD performance requires a data representation that bridges the distributional gap, as well as a solving strategy that uses the model's capabilities specifically for the tasks it was trained to solve.

Contents

Chapter 1

Introduction

The recent rise of Large Language Models (LLMs) has shown the remarkable ability of neural architectures to memorize and compress information. Even though LLMs' zero-shot capabilities are impressive [1], their performance is limited outside of distribution (OOD) where the model was trained[2]. To create new knowledge it is necessary for models to move past this limit, and be able to abstract the principles of a task from given instances and execute them. This characteristics challenge classical machine learning, and have formed a separate field of *Neural Algorithmic Reasoning* (NAR)[3]. The promise is that neural learner can adapt algorithms into the real-world problems automatically, finding a better-fitting solutions than hand-designed algorithms. Such algorithm should also generalize on instances of unseen sizes where the same solving principles apply. This ability to generalize OOD is the distinguishing quality of truly general algorithms, and therefore the north star of NAR.

To measure generalization qualities it is suitable to use tasks which are known to be solvable by an existing algorithm. This offers a comparison to what the neural system could have learned, and is therefore often done with prototypical tasks from computer science, like sorting, represented in the CLRS benchmark [4]. Another suitable venue for this challenge are puzzles: solving principles are always the same but the instances can differ in size. In our work, we choose the Tents puzzle from the PUZZLES benchmark [5], which is a variation of a constraint satisfaction problem. We continue in the same spirit of the previous work on this benchmark [6] to:

- design a data representation that enables generalization,
- quantify the limits of generalization outside of domain related to puzzle difficulty,
- explore the limits of a diffusion transformer on generalization over puzzle difficulty and size,
- use ensemble methods of diffusion models to improve generalization outside of the domain.

Chapter 2

Related Work

Learning of algorithms allows for use of many different machine learning paradigms [3]. Early graph neural networks (GNNs) have received significant attention [7, 8]. Naturally, many algorithmic problems can be encoded as a graph, and this architecture is the best suited one for treating those. More recently, with the growing popularity of LLMs, the focus has turned to testing their solving and generalization abilities. However, those are not able to generalize out of distribution [2]. There are benchmarks which investigate this to understand what is going wrong with our models[9], and generate data that might be able to help with this [10]. One way to teach autoregressive model to execute an algorithm is to teach it to mimic it within its context window. This has proven useful in learning sudoku [11] and chess [12].

Energy-Based Models Alternatively, reasoning can be be understood as as energy minimization problem. However, for that, first an energy landscape over the solution has to learned. This is captured in the paradigm of energy-based models, which learn an energy function to verify solutions. This enables generalization through composition of individual energy function: for example, in a graph one can learn energy of a single edge and then compose those functions for the whole graph [13]. This has been further explored and refined with sampling schemes with attention to composability [14].

Inductive Biases However, there is an inherent trade-off between how general an architecture is, and how much it is adapted to a specific problem. The latter usually enables better generalization on a specific problem, but less so on a different problems. If input and output interfaces of the neural learner are designed well, one can use imitation to learn an algorithm that generalizes perfectly [15]. One can also try to align a neural network to an algorithm by forcing it to keep a trajectory of combination of different fixed steps, which also creates a provably general algorithm [16]. Moreover, the concept of neural compilation directly injects an algorithm into the parameters of a network [17]. However, one might argue that this induces too much bias in the architecture and does not make for

a general learner.

Diffusion Recently, the original diffusion invented for generative imaging [18] has also been used for problem solving, for example of sudoku grids [19]. More importantly, one can consider discrete categorical distributions [20], and adapt diffusion to arbitrary tokens, for example text. This gives promises for more expressive solving capabilities as compared to the autoregressive mechanism giving rise diffusion LLMs [21, 22]. As mentioned previously, the GNN architecture have been popular for solving algorithmic problems, one can see those as approximation of the diffusion process, which gives the promise of good performance and generalization capabilities [23].

Diffusion for Problem Solving To use diffusion for problem solving, one has to define diffusion on graphs. Recent approaches consider variants of discrete Bernoulli noise and continuous Gaussian noise with post-processing of proposed solution to create a valid discrete solution in graph problems [24]. This approach shows possibility of generalization, however, still struggles out of distribution where performance gap to the best found solution by an optimizer increases from 0.01% in distribution to 3.9% outside of distribution. An even more promising approach is presented by [25] which constrains the sampling of the solution space towards solution and using a divide-and-conquer strategy to generalize to new puzzles.

Decomposition A general principle in many works tackling generalization is decomposition. In [26], generalization is achieved by recombining outputs on subsampled subgraphs into a bigger graph, which is then used to generate a solution. Similar decomposition appear also in [25]. Finally, in energy-based models one can exploit decomposition principles too[14]. This hints that one should exploit as much as possible to abilities of the model within the training distribution to find the best outcome outside of training distribution.

Chapter 3

The Puzzle: Tents

In our work we focus on the Tents puzzle from the the PUZZLES Benchmark [5]. The puzzle has simple mechanics: a number is assigned to each row and each column, which indicates how many 'tent' cells should be placed there. Each 'tent' has to be placed next to a 'tree'. No two tents are adjacent, not even diagonally. Sometimes, a tree is adjacent to more than just its own tents, but it is always orthogonally adjacent to its own tent (i.e. sharing a side with it)[1]. The rest of the cells are filled with 'grass'. Importantly, this puzzle features a unique solution.

For illustration, consider the puzzle in Figure 3.1. On the left, we can see the newly generated puzzle, containing only the trees and empty spaces. In the middle, we see puzzle in progress: grass spots are filled, and a couple tents are placed. We can see that two tents that are diagonally adjacent display a conflict. On the right we see the full solution of the puzzle, with all the tents being placed correctly and all empty spots marked as grass.

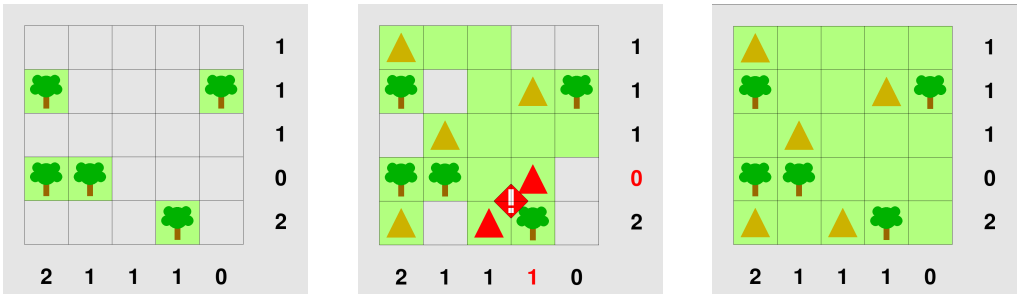
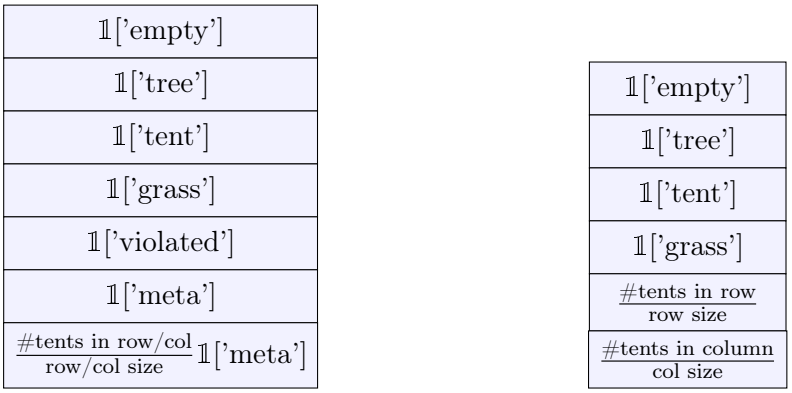


Figure 3.1: Illustration of the tents puzzle. *Left*: Unsolved puzzle. *Middle*: Wrong solution of the puzzle, displaying conflict of two tents being adjacent diagonally. *Right*: Solved puzzle.

Data Representation To represent the data we convert from the representation given by the generator [5]. This produces a graph containing a node for each place in the grid, and additionally meta nodes, which contain the constrains

[1] https://www.chiark.greenend.org.uk/ sgtatham/puzzles/js/tents.html

of the puzzle. Each of the nodes is first encoded one-hot for its type, and additionally, if the node is a meta node, a normalized number of tents per available spots in that row or column is added to the last feature of the node if any. This is visualized in Figure 3.2a. However, to simplify this and enable generalization we copy the related normalized feature from corresponding meta node in a row or a column into the 5th and 6th position in the vector for each node. Hence we get features as presented in Figure 3.2b.



(a) Original Node Feature Encoding (b) New Node Feature Encoding

Figure 3.2: Comparison of new encoding puzzle features. *Left*: The original encoding of the puzzle encoded meta nodes separately, injecting them with the information of the number of tents in a certain row or column they were attached in. In transformer, this representation prevents generalization as the positional encoding of the nodes might not give enough directions for nodes to pay attention to their relevant meta nodes. *Right*: The new representation solves this problem by collocating the information of the meta nodes in the node itself.

This adjustment gives us the opportunity to drop the meta nodes, and organize the remaining nodes into a simple grid, Figure 3.3 shows this. Note how nodes that have been produced by the generator are lined up in the grid, their positions will be used for positional encoding. Finally, by denoting a the size of the grid by S we denote that one datapoint is a tensor x_i of size $S \times S \times F$. The solution to the puzzle is a vector of the correct classes, i.e. $y_i \in \{0, 1, 2, 3\}^{S \times S}$. We denote the dataset of points a $D = \{(x_i, y_i)\}$.

Metrics We note that to solve this puzzle, one can obtain 80% cell accuracy by copying the trees into the output and then predicting everything else as empty. This is trivial for the model to do, therefore it makes more sense to talk about the amount of puzzles which were completely fully solved. This is reflected in the metric called *instance accuracy*, which is described in Equation 3.1. We will

1	2	3	4	5	**31**
6	7	8	9	10	**32**
11	12	13	14	15	**33**
16	17	18	19	20	**34**
21	22	23	24	25	**35**
26	**27**	**28**	**29**	**30**	

5x5 Grid with Meta Nodes

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

5x5 Augmented Grid

Figure 3.3: Comparison of two data representations. *Left:* 5x5 grid with meta nodes marked in green, which influence the respective row or column. *Right:* 5x5 grid where each node holds the meta information of the corresponding meta nodes in a row or column. The representation on the right generalizes better. The positional encoding are used on these grid positions.

be using this metric throughout to describe the performance of approaches.

$$\text{instance\_accuracy}(D) = \frac{1}{|D|} \sum_{i=0}^{|D|} \mathbb{1}\{y_i = \hat{y}_i\} \tag{3.1}$$

Generated Data We generate a training set of 64 000 puzzles of the size 5x5, which we complement with a 8000 validation set and 8000 test set. For testing outside of training distribution, we generate 240 datapoints of size 4x4, and 8000 for each of 6x6 till 10x10. All puzzles are of the 'easy' difficulty. They are all sampled randomly such that the solutions are different for each puzzle.

Table 3.1: Distribution of Generated Puzzle Datasets by Grid Size. Each puzzle is randomly generated and has a unique solution. The size of the dataset is limited by the amount of possible existing puzzles of a certain size: for 5x5 the maximum number of puzzles was generated, and then the dataset was split into validation and evaluation. Subsequently, the same amount of evaluation data was generated in different sizes.

Grid Size	**Training**	**Validation**	**Evaluation**	**Total**
4×4	-	-	240	240
5×5	64,000	8,000	8,000	80,000
6×6	-	-	8,000	8,000
7×7	-	-	8,000	8,000
8×8	-	-	8,000	8,000
9×9	-	-	8,000	8,000
10×10	-	-	8,000	8,000

Chapter 4

Model Architecture

Transformer Encoder For our model, we use the original transformer architecture [27]. The features of the model first get upscaled to 32 dimensions, then get passed through 6 transformer encoder layers. Each have with multihead scaled dot-product attention as stated in Equation 4.1, where all features get upscaled into $d_k = 512$ dimensions and then split into 4 heads for the attention, as described in Equation 4.2. These architectural choices are chosen to make the work comparable with previous work [6]. Finally, the output is projected to number of classes of the puzzle, which in the case of tents is 4. The outputs are treated as logits of predictions of the present classes. For use PyTorch implementation of the transformer encoder[1].

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{4.1}$$

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \\ \text{where head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{4.2}$$

Positional Encoding To encode the 2D structure of the puzzle (as displayed in Figure 3.3, we use Rotary Position Embedding (RoPE), this has proven effective in many settings for speeding up convergence as opposed to sinusoidal positional embeddings, thanks to preserving inner product for same relative position in a sequence [28]. This embedding encodes the position of the vector by treating adjacent pairs of indices as 2D vectors and rotating them based on the position of the word with different frequency for different components. This is then generalized to 2D by cutting up the vector to two parts. We firstly rotate the first part according to the x coordinate of the point, and the second part according to the y coordinate of the point. With hidden dimension 32, we have to rotate two vector of size 16. Each of which is split into pairs, forming 8 2-dimensional vectors. Each frequency to rotate with is then calculate as in Equation 4.3, where

[1] https://docs.pytorch.org/docs/stable/generated/torch.nn.TransformerEncoder.html

we choose $M = 5$ as parameter.

$$f_i = 1 + \frac{Mi}{8}\pi \tag{4.3}$$

Let us then take the embedding on the position (x, y), and consider $(j, j+1), j \equiv 1 \mod 2$. Let us assume that it is in the first half, i.e., $j < 32$. Then the vector will rotate according to its x position with the angle $x \cdot f_{\frac{j+1}{2}}$.

Training Objective For each thing on input we predict the logits for 4 classes, and we use categorical Cross Entropy Loss, which defines loss for each prediction as in Equation 4.4[2].

$$l_n = -w_{y_n} \log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^{C} \exp(x_{n,c})} \mathbb{1}\{y_n \neq \text{MASK}\} \tag{4.4}$$

Here w_{y_n} denotes the weight of the output class. To balance the training, we note that our classes are in the ratio 1:1:3 ('tent':'tree':'empty'), hence we weigh the classes inversely with weights $\frac{5}{1} : \frac{5}{1} : \frac{5}{3}$ additionally normalized to sum to 1.

[2]https://docs.pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

Chapter 5

Background: Discrete Difussion

The architecture described in Chapter 4 has a very simple principle: the model has to guess all cells at the same time. However, this has one big disadvantage: the model cannot leverage partial information by filling in the cells which it is certain about with first, and then spending its resources to predict more difficult cells. In this subsection, we describe how one can augment the transformer model discussed above into a diffusion transformer model.

Principle of Denoising Diffusion Diffusion models aim to model the reverse process $p_\theta(x_{0:T})$, which is the inverse of the forward process $q(x_{1:T}|x_0)$, which models a probabilistic addition of noise to the data [18]. As a result, the reverse process can be used for generation of samples in a target distribution purely from sampled noise. Usually, for continuous transition the noise is Gaussian, however, for solving puzzles we turn to the generalization to the discrete case [20] on which we base this analysis. In the following analysis, we focus on one puzzle cell at the time, and we treat those interdependently. The information about the surrounding cells is shared using the attention mechanism in the transformer. The forward process is defined by transformation of probability mass by the matrix Q_t:

$$q(x_t|x_{t-1}) = \text{Cat}(x_t; x_{t-1}Q_t) \tag{5.1}$$

where Cat denotes a categorical distribution on hot encodings of a class encoding of the class. The matrix Q_t then transitions the density between the states. Due to Markov assumption, the distribution density is therefore tractable at every step t through chaining of matrices Q_t, setting $\prod_{i=1}^{t} Q_t = \bar{Q}_t$:

$$q(x_t|x_0) = \text{Cat}(x_t, x_0 \prod_{i=1}^{t} Q_t) = \text{Cat}(x_t, x_0\bar{Q}_t) \tag{5.2}$$

As stated above, for solving the puzzle we are interested in the reverse process, for which we get by Bayes' Rule:

$$q(x_{t-1}|x_t, x_0) = \frac{q(x_t|x_{t-1}, x_0)q(x_{t-1}|x_0)}{q(x_t|x_0)} \tag{5.3}$$

Now we note that

- $q(x_{t-1}|x_0) = x_0\bar{Q}_{t-1}$ as per Equation 5.2,
- $q(x_t|x_{t-1}, x_0) = q(x_t|x_{t-1})$ using Markov property. Interpretin vector x_t as one hot on the state, the probability of tranfering there from the previous step is exactly $x_t Q_t^\top$
- For normalization we take the probability of entering being in the state x_t under the current probability distribution $x_0\bar{Q}_t$, which can be selected by the one hot vector as $x_0\bar{Q}_t x_t^\top$.

This gives use that, using element-wise multiplication in the numerator:

$$\frac{q(x_t|x_{t-1}, x_0)q(x_{t-1}|x_0)}{q(x_t|x_0)} = \text{Cat}\left(x_{t-1}; p = \frac{x_t Q_t^\top \odot x_0\bar{Q}_{t-1}}{x_0\bar{Q}_t x_t^\top}\right) \tag{5.4}$$

Inference with Diffusion Similarly to [18, 29], instead of predicting directly $p_\theta(x_{t-1}|x_t)$ we instead predict $p_\theta(x_0|x_t)$, and then marginalize using Equation 5.4. We note that the term $q(x_{t-1}, x_t|\tilde{x}_0)$ is fully tractable.

$$p_\theta(x_{t-1}|x_t) \propto \sum_{\tilde{x}_0} q(x_{t-1}, x_t|\tilde{x}_0)\tilde{p}_\theta(\tilde{x}_0|x_t) \tag{5.5}$$

For our use case, we only model transition of tokens between the solved state ('tent' or 'grass') and the masked state ('empty'), which is equivalent to absorbing-state diffusion in [20]. This means we can summarize our transitions for q as:

- If state is 'empty' it stays empty.
- If state is either 'tent' or 'grass', it changes to 'empty' with probability p_s, and stays the same with probability $1 - p_s$.

This can be summarized by the matrix Q below:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ p_s & 0 & 1-p_s & 0 \\ p_s & 0 & 0 & 1-p_s \end{bmatrix} \tag{5.6}$$

Now if we evaluate $q(x_{t-1}|x_t, x_0)$ for transition from the 'empty' state to either 'grass' or 'tent' state, we observe that we have using Equation 5.4, taking $i \in \{2, 3\}$:

$$q(x_{t-1}|x_t, x_0)[i] = \frac{x_t Q_t^\top[i] \cdot x_0\bar{Q}_{t-1}[i]}{x_0\bar{Q}_t x_t^\top} = \frac{p_s\frac{1}{4}(1-p_s)^{t-1}}{x_0\bar{Q}_t x_t^\top} \tag{5.7}$$

We see that this is the same independent of i, therefore if we had to 'denoise' a certain field in the puzzle using the information from $\tilde{p}_\theta(\tilde{x}_0|x_t)$ in Equation 5.5 and we had to choose between 'tree' and 'grass', we see that based on q the probabilities are proportional. Therefore, we can (up to normalization) just rely on the probabilities given by $\tilde{p}_\theta(\tilde{x}_0|x_t)$. In the end, when we have to sample a discrete distribution, we have to choose for one of those classes. In that moment, the probabilities of picking the respective classes are proportional to the probabilities given in p_θ. To use the information provided by this distribution to its maximum potential, we choose to instead of sampling the distribution q and combining it with $\tilde{p}_\theta$, we choose to take the consider the probability distributions of all tokens (as they are sampled independently anyway) and take the one which has maximum probability. This empirically works better then probabilistic decoding on solving task.

Training Objective For simplification of the training objective, we turn to analysis in [30]. We consider $q(z_t|x) = \text{Cat}(z_t; \alpha_t x + (1-\alpha_t)\text{MASK})$, meaning α_t regulates how much noise we add to the data. Therefore, we add noise to any 'grass' or 'tent' cell with probability $1-(1-t) = t$, $t \in [0,1]$. Let $\alpha'_t = \frac{\partial \alpha_t}{\partial t}$. We consider the linear scheduler:

$$\alpha_t = 1 - t \tag{5.8}$$

$$\alpha'_t = -1 \tag{5.9}$$

Then using $\frac{\alpha'_t}{1-\alpha_t} = \frac{-1}{1-(1-t)} = \frac{-1}{t}$ the training NELBO objective from Equation 10 in [30] where expectation over q signifies corruption of data into z_t takes the form:

$$\mathcal{L}^{\infty}_{\text{NELBO}} = \mathbb{E}_q \int_{t=0}^{t=1} \frac{\alpha'_t}{1-\alpha_t} \log\langle x_\theta(z_t, t), x\rangle \, dt \tag{5.10}$$

$$= \mathbb{E}_q \int_{t=0}^{t=1} \frac{-1}{t} \log\langle x_\theta(z_t, t), x\rangle \, dt \tag{5.11}$$

The expectation over q in this equation yields the appropriate Minimization of this objective matches up to class weighting and scaling by $1/t$ the objective set out in Equation 4.4. Note that this is a critical part of modification of the training, to show this we dedicate the Section A.3.

Chapter 6

Methods

To check the generalization of our model we need to clarify the training conditions of the model, the data that they will be using and the inference conditions. We start by discussing the data and specific of the optimization. We later describe the specific models and their inference models.

Data Corruption As described in Chapter 5, our noise comes from discrete distribution, masking the data towards a distribution of fully masked tokens. This simply means that during each part of the process, the nodes can transition from 'grass' or 'tent' to the 'empty' state. Note that we do not add noise to the trees, as that would make the puzzle unsolvable. For the same reason, if cell has noise added to it, we do not remove the information from the meta nodes about the number of tents in a row / column. We refer to the process of adding noise as 'corruption'. If we say we corrupt a puzzle with a certain probability c, it means that each cell is turned into 'empty' independently with this probability. Note that during training, we ignore coming from the points that have already been solved. Those are then simply copied over in the solution during inference.

Positional Encodings For generalization onto different size, in our case from 4x4 to 10x10, it would be problematic if the model has never seen the different frequencies with which points further on the edge of the puzzle were rotated with. For example, in 6x6 the last pair of vectors would be rotated further than in 5x5 puzzle. In order for the model to generalize, we apply the positional encoding from a random 5x5 subgrid of the 10x10. Note that this really helps generalization onto bigger puzzle size, as otherwise the model is not able to function when increasing the size of the puzzle. This is an important modification which enhances the performance of the model, we observe the impact of this in Section A.4.

Optimization Settings For all of our models we use AdamW [31] with learning rate 10^{-4} and weight decay 10^{-4}. Although we have found that constant learning rate leads to fast convergence (within 100-150 epochs), the generalization results

have been suboptimal. Instead, we use one cycle learning rate [32], which quickly ramps up the learning rate and then cools it down. We train until convergence on the in-distribution validation data. Using batch size 512, till maximum 10000 epochs with patience of 1000 on delta of 10^{-5}. However, the training ends usually ends around 2500 epochs.

6.1 Encoder Model

The simplest model in our setup is referred to as *encoder model*. We trained the architecture specified in Chapter 4 on puzzles of size 5x5 on specific level of corruption c. During testing we evaluate this model on test set outside of distribution of size 4x4 till 10x10 with the same corruption level c.

6.2 Diffusion Model

To turn our transformer model to a diffusion model, we use the reweighting of the terms in the loss as in Equation 5.11. We multiply the factor $1/t$ with the class weighting factor for each term. We obtain an objective proportional to the integral in Equation 5.11 by uniformly sampling $t \in [0, 1]$ and corrupting in each at that step with that probability. We refer to the model trained using this objective as *diffusion model*. On inference we can choose to let the model one-shot solve the puzzle with $\tilde{p}_\theta(\tilde{x}_0|x_t)$, this we refer to as *diffusion model* with *no decoding*.

However, as this model is trained for diffusion, we can use it to decode step by step. At each step, we use $\tilde{p}_\theta(\tilde{x}_0|x_t)$ to predict the probabilities for each token, and then take the token that has the highest probability. We insert that class into the representation and run the next step, until the puzzle is solved. We refer to this as *diffusion model with stepwise decoding*.

6.3 Ensemble Model

Following the example of [26], we would like to explore the compositional usage of the diffusion model for solving bigger puzzle instances. We would like to leverage the abilities of the model which has been trained on 5x5 puzzles, by cutting the bigger puzzles into 5x5 grids and composing the predictions together.

We do this by running the inference separately for each of the grids and averaging the probabilities of the predicted classes. This gives us a new estimate of the probabilities for each of the classes, we apply those classes using the same method as for *diffusion model with stepwise decoding*. This is visualized in Figure

6.1. In the results, we refer to the model using this decoding as the *ensemble model*.

Furthermore, it is also possible to average the logits, and each has its advantages [33]. However, in our case we do not see a notable difference, as discussed in Section A.2. Note that the are of ensembling multiple diffusion models is fairly unexplored area. For example in visual tasks, even though the model scores better, the fidelity is not improved when multiple models are involved [34].

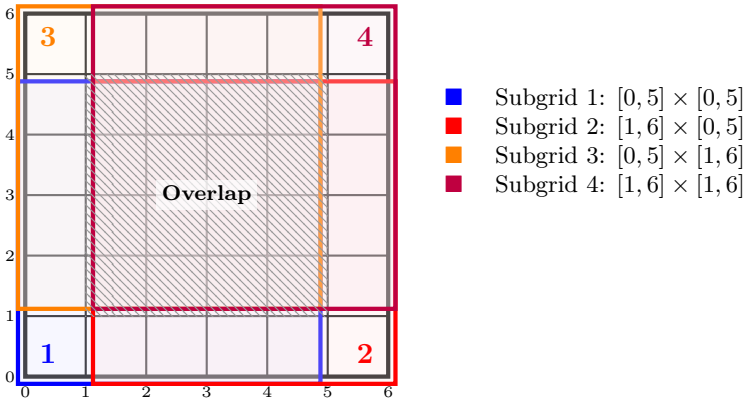


Figure 6.1: Visualization of the ensemble model decoding process for a 6×6 puzzle. The larger grid is decomposed into four overlapping 5×5 subgrids. Predictions from the diffusion model are computed for each subgrid, and class probabilities are averaged in the overlapping regions (in this case the central 4×4 area) to produce the final global prediction.

Chapter 7

Experiments & Results

7.1 Noise-Dependent Generalization

Motivation Firstly, we would like to relate the difficulty of the solved puzzled and the ability of the model to generalize. We would like to check whether there is an interesting behavior with respect to how the difficulty changes. We observe that if a puzzle is 0% corrupted (i.e. the puzzle is solved), then the task of the model is to basically copy a one-hot encoding onto the logits for each token individually. As long as the positional encoding does not perturb the information outside of the training distribution, this should be a simple task. In contrast, solving the whole puzzle in one-shot with 100% corruption is difficult, as it requires many computational steps. Additionally, extending outside of distribution means that the model has to integrate more logical steps and pay attention to more tokens.

Experiment Setup To investigate the relation of difficulty of corruption, we train two *encoder models*s, one in puzzles with 0% and 100% corruption. We evaluate both models on 6x6 puzzles of corresponding corruption. The results are visible in Figure 7.1.

Results The result in Figure 7.1 indicate that it is very challenging for the model to learn to solve puzzles of higher size, as it does not manage to solve almost any puzzles in 6x6 for 100% corruption. With our setup, we can continuously vary the percentage of corrupted parts of the puzzle. To explore the performance of different models, we include the generalization ability of models trained on $T = 5$x5 and evaluated on sizes until 10x10 with noise level $0\%, 25\%, 50\%, 75\%, 100\%$. The results are visible in Figure 7.2. We observe that with increasing level of noise, the out-of-distribution generalization becomes harder and harder. Eventually, for 100% corruption, we see almost no OOD generalization.

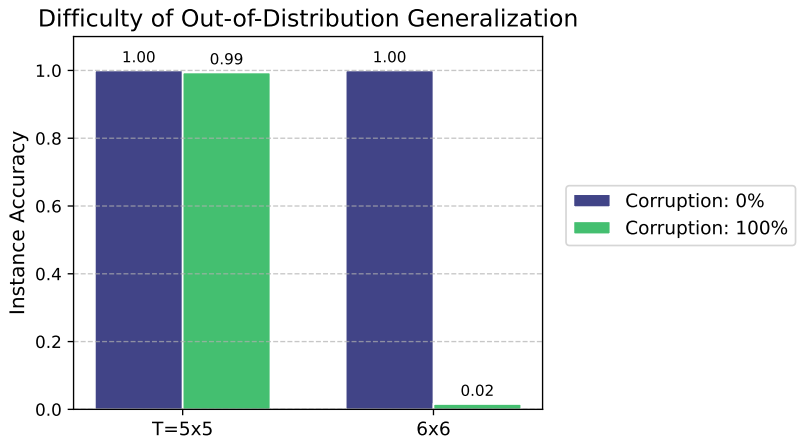


Figure 7.1: Performance of models outside of their training distribution. *Encoder model* was trained on 5x5 puzzles and evaluated on 6x6 puzzles, for 0% and 100% corruption respectively. The results on the testing set show the difference in ability to generalize on a trivial task, but lack of generalization on the difficult task of solving the puzzle from scratch.

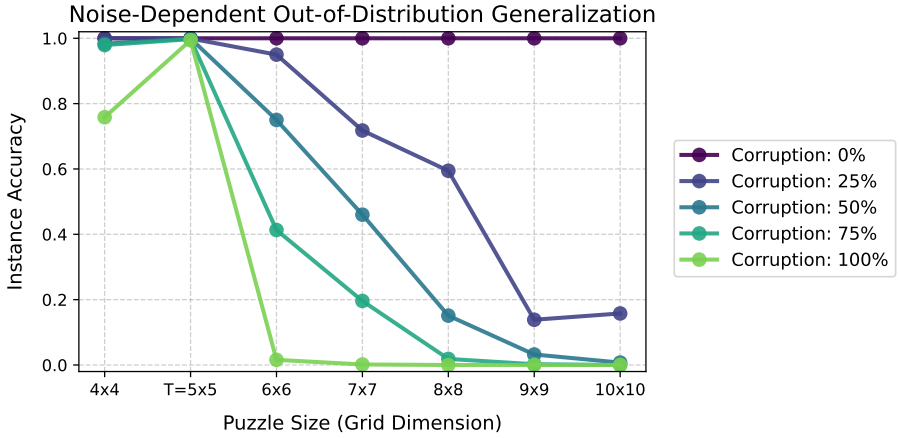


Figure 7.2: Generalization on noise levels. The *encoder model* has been trained on 5x5 puzzles with varying levels of noise and evaluated outside of the training distribution. Each of the puzzles is corrupted to the level specified by the legend, and the testing data held equal corruption rate. Generally, the out-of-distribution generalization is less successful the higher the noise in the data is.

7.2 Decoding Generalization

Motivation & Setup Secondly, we would like to investigate the ability to generalize for the diffusion models. We note that this model has been trained on uniformly corrupted puzzles of size 5x5. We first investigate the ability of the model to solve puzzles of sizes 4x4 till 10x10, to get a better insight into what is happening, we once again look at the puzzles at different corruption levels. The results are presented in Figure 7.3.

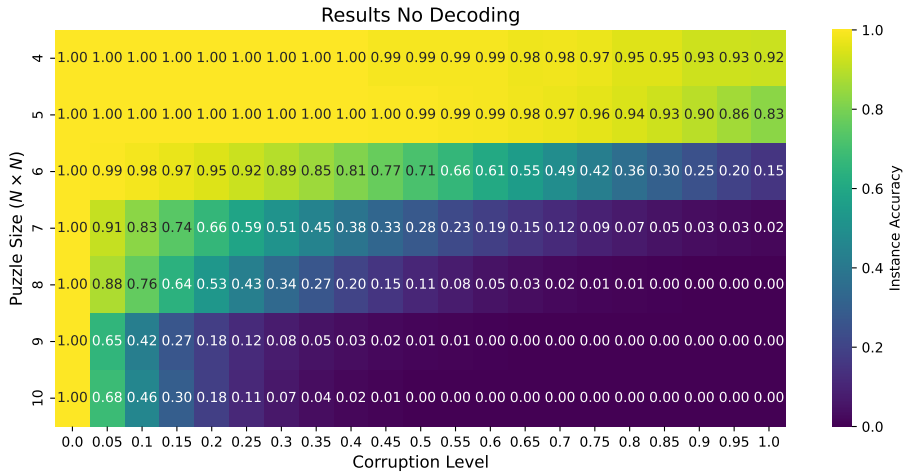


Figure 7.3: Performance of *diffusion model* with *no* decoding across puzzle sizes. The model was trained on 5x5 data of uniform corruption levels. We observe that in distribution and on smaller puzzle sizes (4x4) the model obtains a good performance. However, moving even slightly outside of the domain yields a big decrease in the model performance.

No Decoding Results We see that for sizes 4x4 and 5x5, the models manage to solve almost all difficulties of puzzles. However, with increasing size beyond 6x6 the ability to solve puzzles quickly drops. For puzzles of size 9x9 and 10x10 the model is not able to solve any of the fully corrupted puzzles.

Decoding Results As discussed in Section 5, this model is supposed to be used with stepwise decoding in inference. We first perform the same analysis as with *no decoding*, showing the performance of the model across the size. To better understand on which puzzles it does better we also visualize the difference of the performance of the two methods. Both results are visible of the decoding model for all sizes are visible in Figure 7.4. We can clearly see that the decoding improves performance, especially when we move right outside of the training

distribution, yielding up to 45% improvement on 6x6 puzzles. However, the gains further out of the distribution are quite modest.

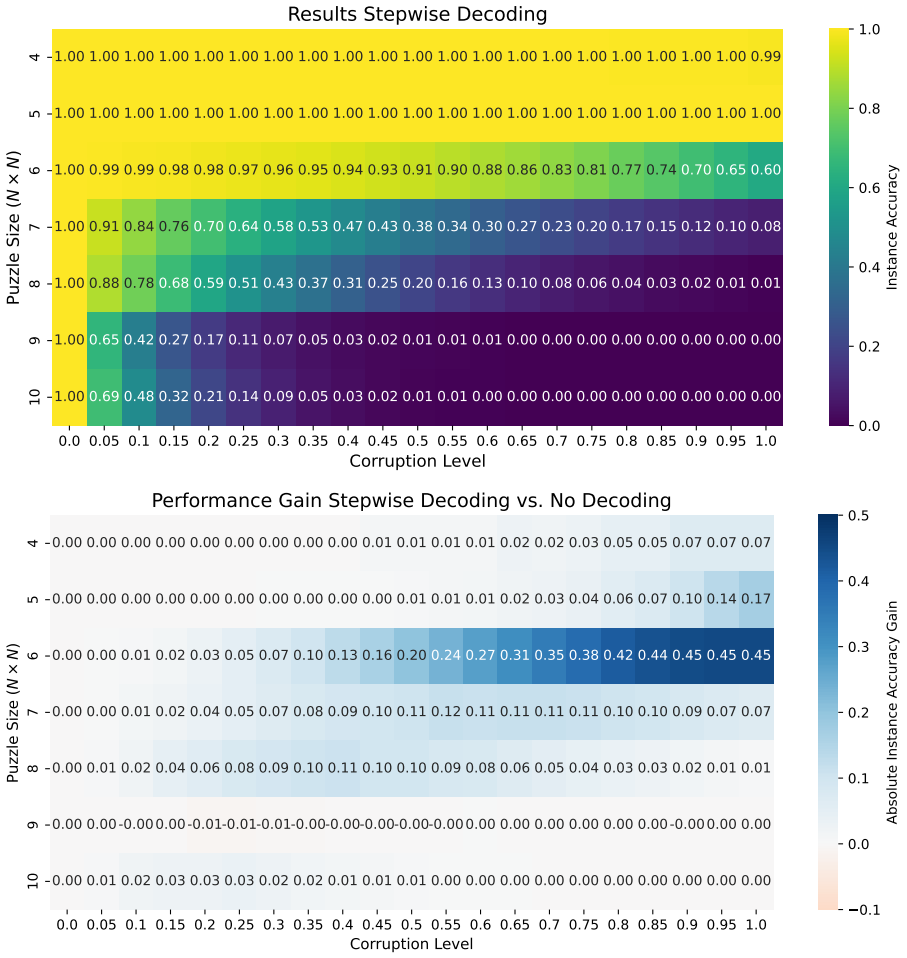


Figure 7.4: Performance of the *diffusion model with decoding* which was trained on 5x5 data of uniform corruption. *Top*: The absolute performance of the decoding model. *Bottom*: The relative gain for using the decoding model as compared to *diffusion model* without decoding as presented in Figure 7.3. The decoding helps the model boost its performance to perfection in distribution, and also yields big improvements right outside of distribution, especially on 6x6 puzzles.

Analysis To better understand what difference the decoding method makes over no decoding we want to inspect the confusion between the classes during the different inference methods in the settings where the difference is the biggest. We plot the confusion matrix at the 6x6 puzzle with 85% noise, where there is a gain of 44% in puzzle instance accuracy. The matrices are displayed in Figure 7.5. We see that *decoding* process reduces the amount of 'grass' that was predicted

to be 'tent'. This means the model predicts too many tents on the 6x6 puzzle. With the *decoding* method, the confusion matrix is quite even on the confusion of the two classes, meaning that this effectively helps to reduce the false positive tents that were predicted. We hypothesize that is caused by the model initially thinking that there are many possible cells, however, as it solves step by step, it manages to eliminate some of those wrong placements.

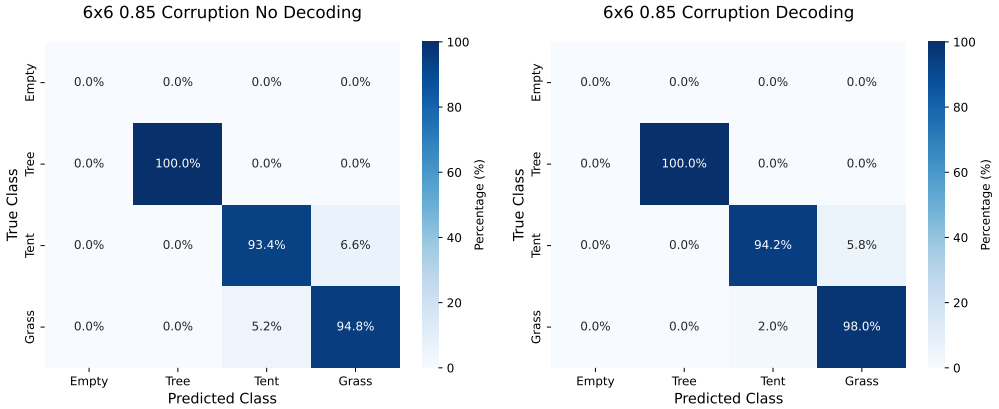


Figure 7.5: Confusion matrices for different decoding method on 6x6 puzzle with 0.85 percent corruption. The decoding method displays a gain of 44% in this area. *Left*: Confusion matrix for no decoding. *Right*: Confusion matrix for stepwise decoding. The decoding method helps the model get rid of many predicted tents, which were supposed to be grass.

.

7.3 Diffusion Ensemble

Motivation As we have seen, purely decoding does not help us scale outside of distribution. This leads to the proposal of *ensemble model* where we would like to enhance the decoding by combining multiple predictions of the model on a bigger grid by averaging the probabilities. We used the same setup as above and execute the same analysis. The results are visible in Figure 7.6.

Results Figure 7.6 clearly displays a clear contribution to the out of distribution generalization. We can see that it performs perfectly on the sizes 4x4 and 5x5, additionally yielding solid improvements outside of distribution for all sizes from 6x6 till 10x10. There seems to be a slight degradation of performance for 6x6 in the middle of the noise distribution. Overall, the *ensemble model* seems to be doing the best right at the difficulty of puzzles where the *no decoding* performance starts to drop fast. This is as expected, as these are the difficulty and size setups where the trained model contains enough information to execute the task, but needs additional information to fully solve the puzzle.

Analysis Similarly to before, to better understand what difference the decoding method makes over no decoding we want to inspect the confusion between the classes. We plot the confusion matrix at the 9x9 puzzle with 15% noise, where there is a gain of 33% in puzzle instance accuracy for the *ensemble model* over *no decoding*. The matrices are displayed in Figure 7.8. In contrast to what happens with the *diffusion model with decoding*, we see that *ensemble model* process reduces the amount of 'tents' that was predicted to be 'grass'. It seems that as we move to bigger sizes of the puzzle, the normalization in the meta information about the columns is less pronounced, leading the model to predict the fewer tents that it should. The ensembling method effectively fixes this. This is exactly the opposite effect as to what we have seen in Figure 7.5, which explains why this method does so poorly on 6x6 puzzles where the *diffusion model with decoding* was dominating. This is confirmed by the slight increase amount of false positive tents in the predictions.

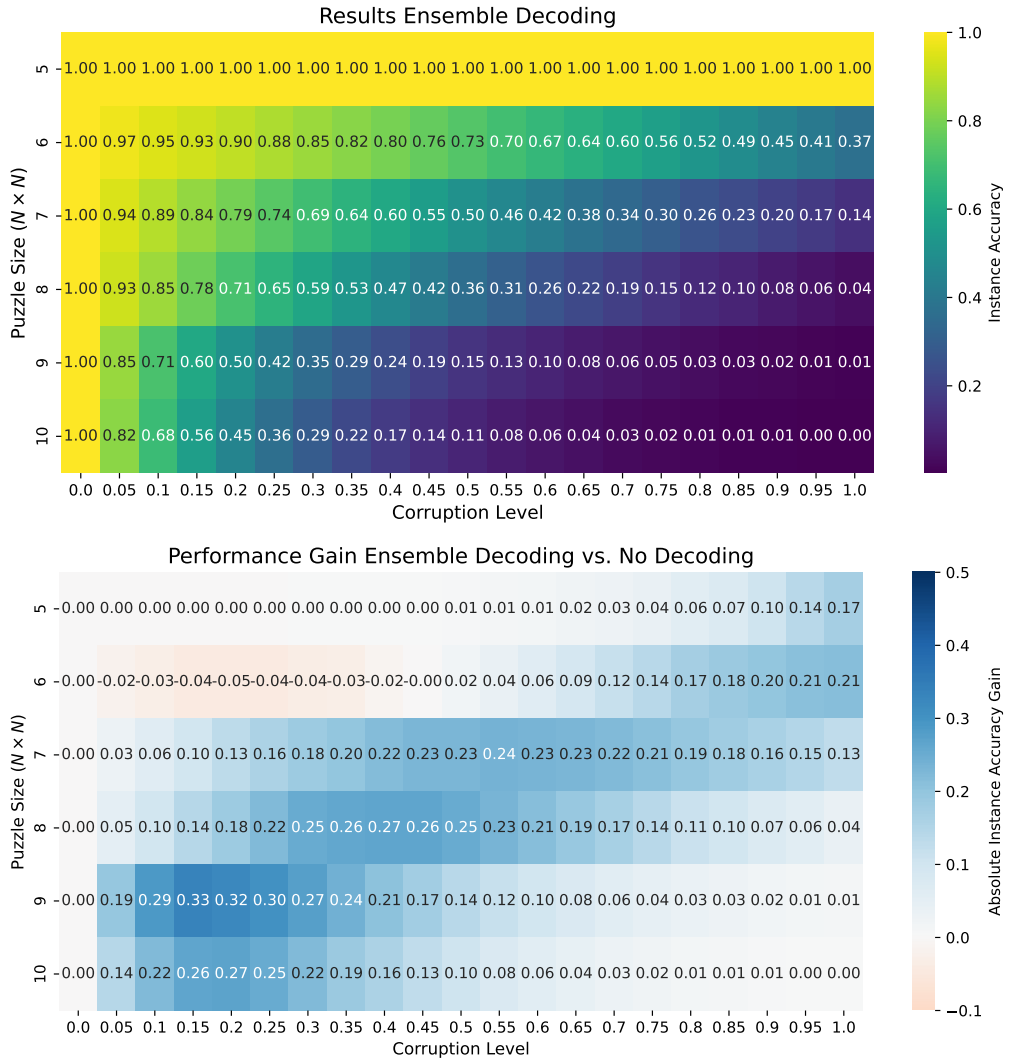


Figure 7.6: Performance of the *ensemble model* which was trained on 5x5 data of uniform corruption. *Top*: The absolute performance of the ensemble model. *Bottom*: The relative gain for using the ensemble model as compared to *diffusion model* without decoding as presented in Figure 7.3. The ensembling helps the model boost its performance to perfection in distribution, but also yields consistent improvements outside of distribution for all sizes

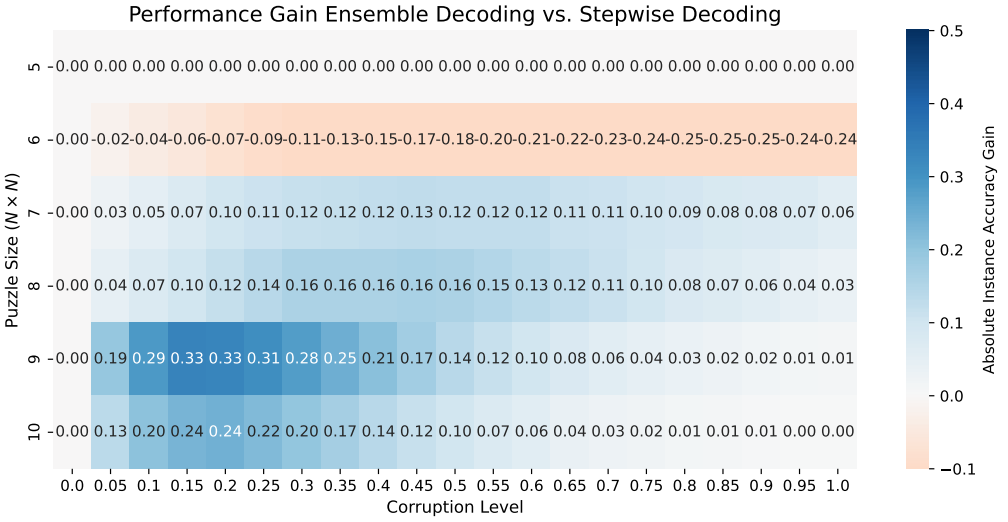


Figure 7.7: Performance comparison of *ensemble model* to *diffusion model with stepwise decoding*. Although the *diffusion model with stepwise decoding* augments the performance right outside of training distribution, but presents only small gains on other sizes, the *ensemble model* consistently performs better on all sizes, giving a good contribution especially far outside of distribution.

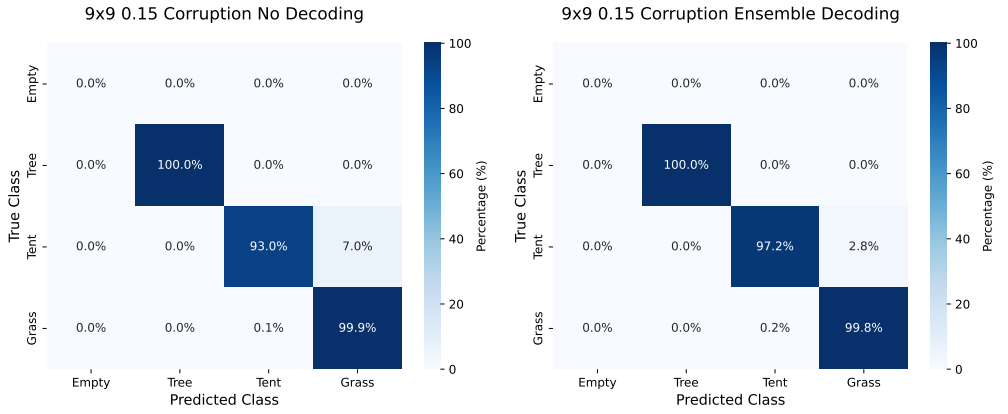


Figure 7.8: Confusion matrices for different decoding method on 9x9 puzzle with 0.15 percent corruption. The ensemble model displays a gain of 33% in this area. *Left*: Confusion matrix for no decoding. *Right*: Confusion matrix for ensemble decoding. The ensemble method helps the model get rid of many predicted grass predictions that should be tents.

7.4 Solution Trajectories

Motivation To understand how models solve the puzzles during *stepwise decoding* for both *diffusion model* and *ensemble model* we observe the tokens that the models predict as the solution progresses. We can observe the confidence by noting the probability of the placed token. To the same plot we also plot the overall accuracy as the model progresses towards the solution. We compare the methods in the previously investigated settings, 6x6 with 0.85 corruption, which can be seen in Figure 7.9, and 9x9 at 0.15 corruption, which can be seen in Figure 7.10.

Results In both settings, the models start by placing the 'grass' tokens as the model is more confident on those. However, as the solutions progress. Th ensemble model quickly starts placing more tents. This aligns with the analysis made above, that the ensembling method prefers to predict more tents. However, more importantly towards the end where most mistakes are committed by both models, the ensemble models lowers the probabilities of the predicted tokens. This could be just that the probability scores are dilluted by averaging over wrongly informed grids, however, one could argue that the probability estimates are better for ensemble model.

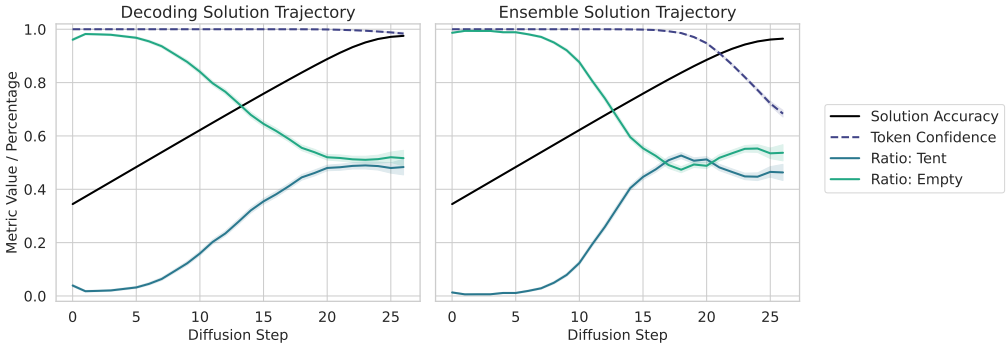


Figure 7.9: Comparison of solution trajectories on 6x6 puzzles with 0.85 corruption. Shaded area is the 95% confidence interval. *Left*: The *diffusion model* with *decoding* method. *Right*: The *ensemble model*. The ensemble model reaches a lower accuracy, however, the confidence scores in its prediction reflect the mistakes. This cannot be said about the confidences of the *diffusion model* with decoding. The *ensemble model* also predicts more tents early on, which confirms the findings in Figure 7.8.

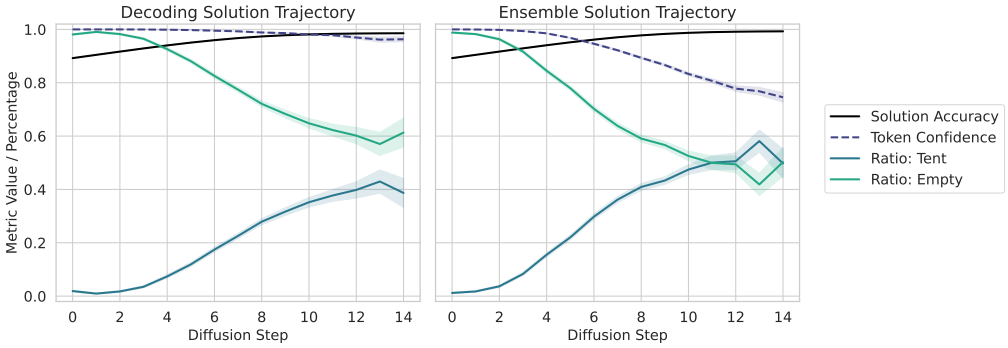


Figure 7.10: Comparison of solution trajectories on 9x9 puzzles with 0.15 corruption. Shaded area is the 95% confidence interval. *Left*: The *diffustion model* with *decoding* method. *Right*: The *ensemble model*. The ensemble model reaches a lower accuracy, however, the confidence scores in its prediction reflect the mistakes. This cannot be said about the confidences of the *diffusion model* with decoding. The *ensemble model* also predicts more tents early on, which confirms the findings in Figure 7.8.

Chapter 8

Discussion & Conclusions

Conclusions In this work, we investigated the ability of simple transformer models and diffusion models to generalize across sizes and difficulty of puzzles. For simple transformer models, we have shown that the ability to generalize is related to the difficulty of the task, with simple task like copying being able to generalize 100%, while solving complex puzzle does not generalize at all with transition in between. Subsequently, for diffusion models we have investigated the ability to generalize across size and difficulty. We have shown that iterative solving process using diffusion yields better results that just trying to one shot the puzzle, even though that is what the model is trained for. The gains in performance, however, were quite modest especially further outside of the distribution. All of this was enabled by change of data representation by injecting the constraint information to nodes themselves. Additionally, positional encodings were randomized during training to improve generalization. Finally, we have investigated ensembling strategy for diffusion models. Using the same model as before, we have cut up the grid to bring inference into distribution. We have shown that this yields promising results in increase of solving accuracy. We have also shown that the probability estimates coming from this model show more realistic decrease in confidence as the solving process of the puzzle continues. Overall, that method decreases the amount of missclasifications and increases generalization across all sizes. This shows that composition of diffusion models to estimate the posterior probabilities can be beneficial, a topic that has not been explored much in problem solving.

Flaws in Methodology Although our results are promising, there are a couple things that certainly deserve attention. Firstly, during this work we have focused only on one puzzle – the tents – and we have refined that data representation to work well for our model. We have not validated the conclusions on other puzzles. This is vital to confirm the conclusions, especially about the efficacy of the ensembling approaches. For this, we would also need to pick puzzles that support the same compositional structure, there are many candidates in the collection of PUZZLES [5], examples being Light Up, Mosaic, Loopy, or Unruly (also used in

[6]). Secondly, due to time constraints, most hyperparameters used in the results were tuned one by one by seeing what works the best. This is a result of the development of research objectives during the experimentation phase. However, once the architecture is fixed, a proper hyperparameter finetuning should be done to find the single set of optimal parameter for each method. This could potentially change the conclusions of this work. Finally, overall a more disciplined approach when comparing to baselines should be taken. Due to the puzzle being quite easy to solve, it would be interesting how models that predict everything as 'empty' and then randomly turn appropriate number of nodes to 'tent' solve a significant amount of the puzzles. Furthermore, from previous work it seems that transformer models perform quite poorly compared to reinforcement learning approaches specified in [6]. Same data representation should be used for both to get a sense of the performance comparison.

Problem Definition Fundamentally, generalization seems to be a data representation problem. To fully enable extrapolation, it might be necessary to further adjust the data format. This is especially true for the meta information about number of tents in rows and columns. This information is normalized in the nodes, therefore for 5x5 puzzle the number is 1 if there is 1 node all free spots are covered in the row, which may refer to different number of trees than which the value 1 would represent on a 6x6. What is more, the topology of the grid was encoded the topology using RoPE. This might be too weak, maybe the encoding as a graph might be better, then confirmed methods for graph diffusion could have been used[25, 24]. Additionally, the diffusion definition we have chosen uses a simple transition from any state to the masked state, which is described by the transition matrix in Equation 5.6. Maybe a better performance could be achieved upon allowing transitions between solved states ('grass' and 'tent'). This approach would also enable us to futher refine a fully solved puzzle, holding a promise of a better solving accuracy.

Theoretical Analysis of Ensembles From a theoretical perspective, the ensembling approaches should be better explored and understood in the context of puzzles. The results are promising, but as of now, there is only little evidence why that would be. Moreover, the averaging of probabilities, though valid approach was not fully justified. There are other possible choices, like averaging the logits, or taking dilated parts of the puzzle, which should be explored and discussed. Overall, the idea of composition seems promising, however, does not work to the extent in the works it was inspired by [26]. Therefore, it should be well understood what are the specifics of the problems that make these techniques work.

Appendix A

Additional Results

A.1 Confusion Matrices and Solution Trajectories of 5x5 Puzzles

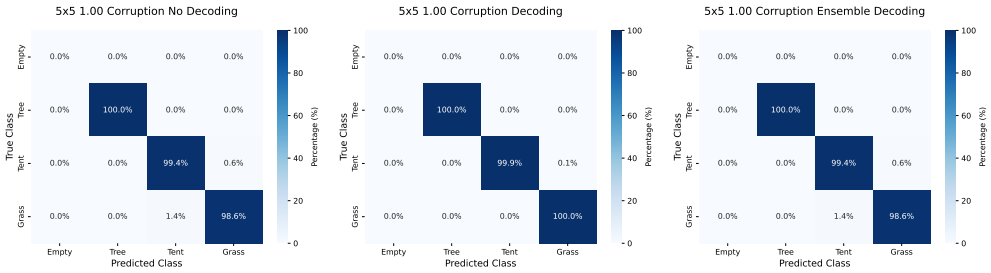


Figure A.1: Confusion matrices for different decoding method on 5x5 puzzle with 1.00 percent corruption. *Left*: Confusion matrix for *diffusion model* with *no decoding*. *Center*: Confusion matrix for *diffusion model* with *decoding*. *Right*: Confusion matrix for *ensemble model*. The *diffusion model* with *decoding* is performing the best, attaining almost perfect scores.

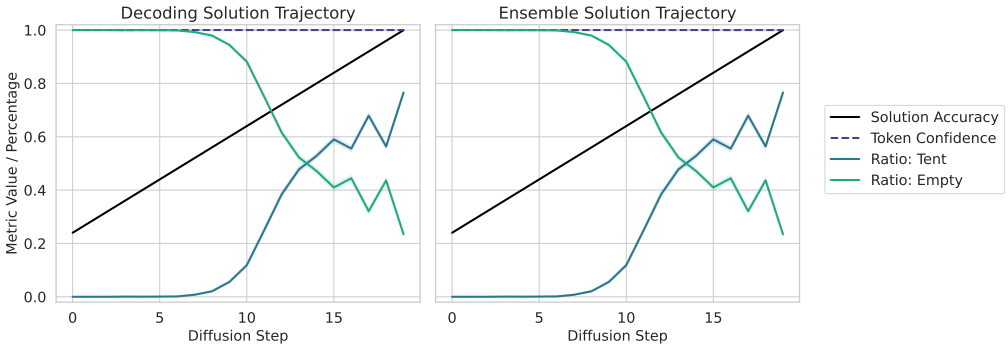


Figure A.2: Comparison of solution trajectories on 5x5 puzzles with 1.00 corruption. Shaded area is the 95% confidence interval. *Left*: The *diffustion model* with *decoding* method. *Right*: The *ensemble model*. The results show exactly same solution trajectories.

A.2 Ensemble Model: Logit Averaging

One particular design choice for the ensemble model was to average the output probabilities instead of averaging the logits. Here we ran the same evaluation but averaging the logits before taking the softmax. The results are presented in Figure A.3. If we compare this to Figure 7.6, we see that there is no substantial difference in two methods.

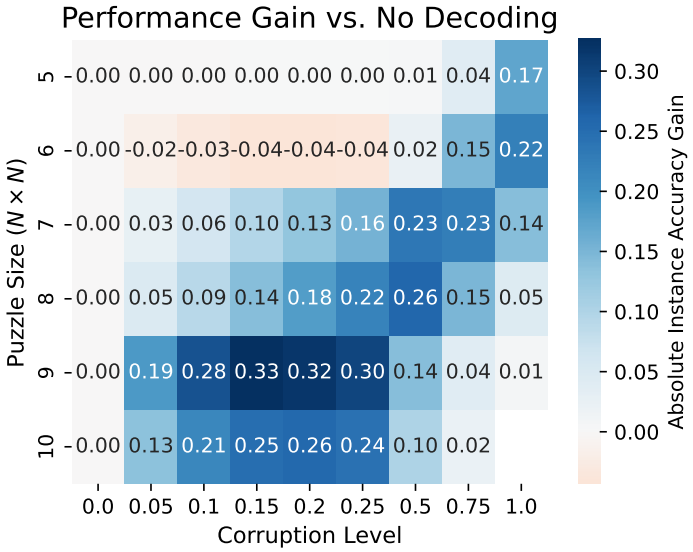
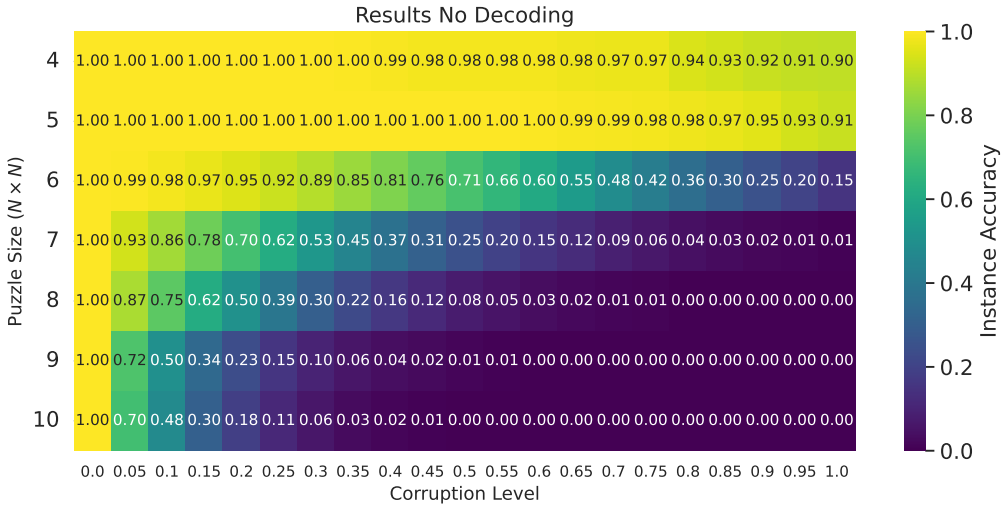
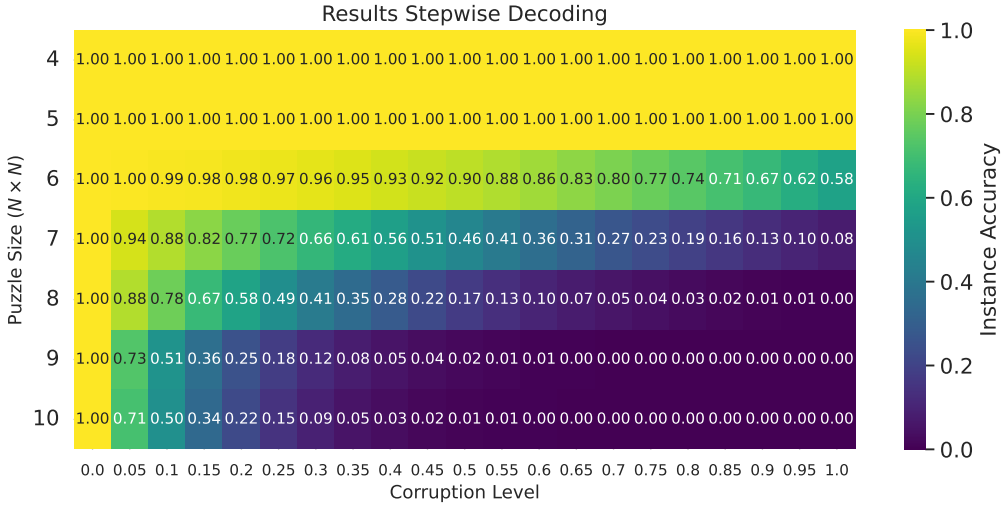


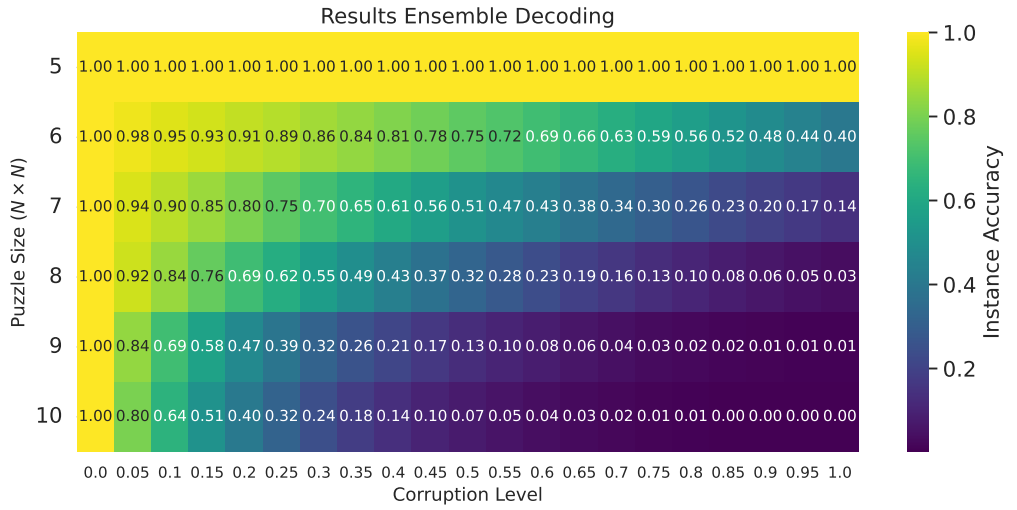
Figure A.3: Performance of the *ensemble model* which was trained on 5x5 data of uniform corruption. The figure displays the relative gain for using the *ensemble model* with mean over logits instead of mean over probabilities as compared to *diffusion model* without decoding as presented in Figure 7.3. This figure is a comparison to the Figure 7.6, we conclude that there is no substantial difference in taking mean over the logits and over the probabilities.

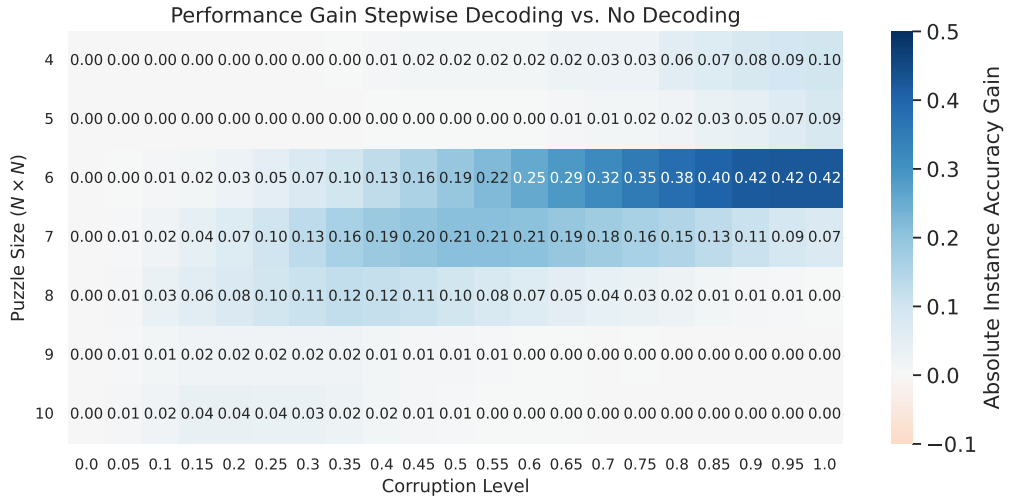
A.3 Loss Scaling

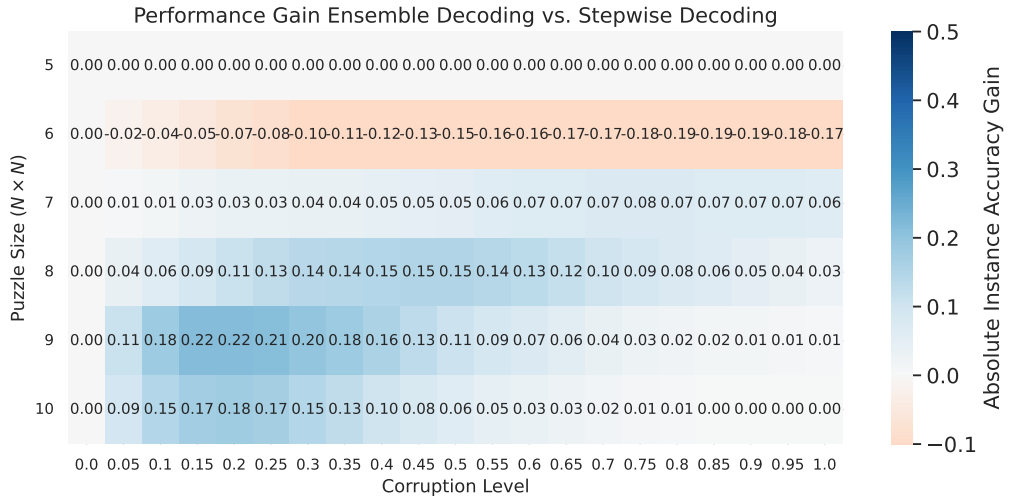
To fit the diffusion loss formulation as presented in Equation 5.11 originally coming from [30] we need to scale our loss by a factor $\frac{1}{t}$. To ablate whether this is beneficial, we train a separate model with a different objective and we run the same suite of test as before. We present the equivalents of the Figures discussed above for this model below. We do not provide commentary as they are just analogues of the previously stated figures. We observe a decrease of performance for both *ensemble model* and *diffusion model* with *decoding*. This is as expected, as the loss now does not follow the proper diffusion objective.





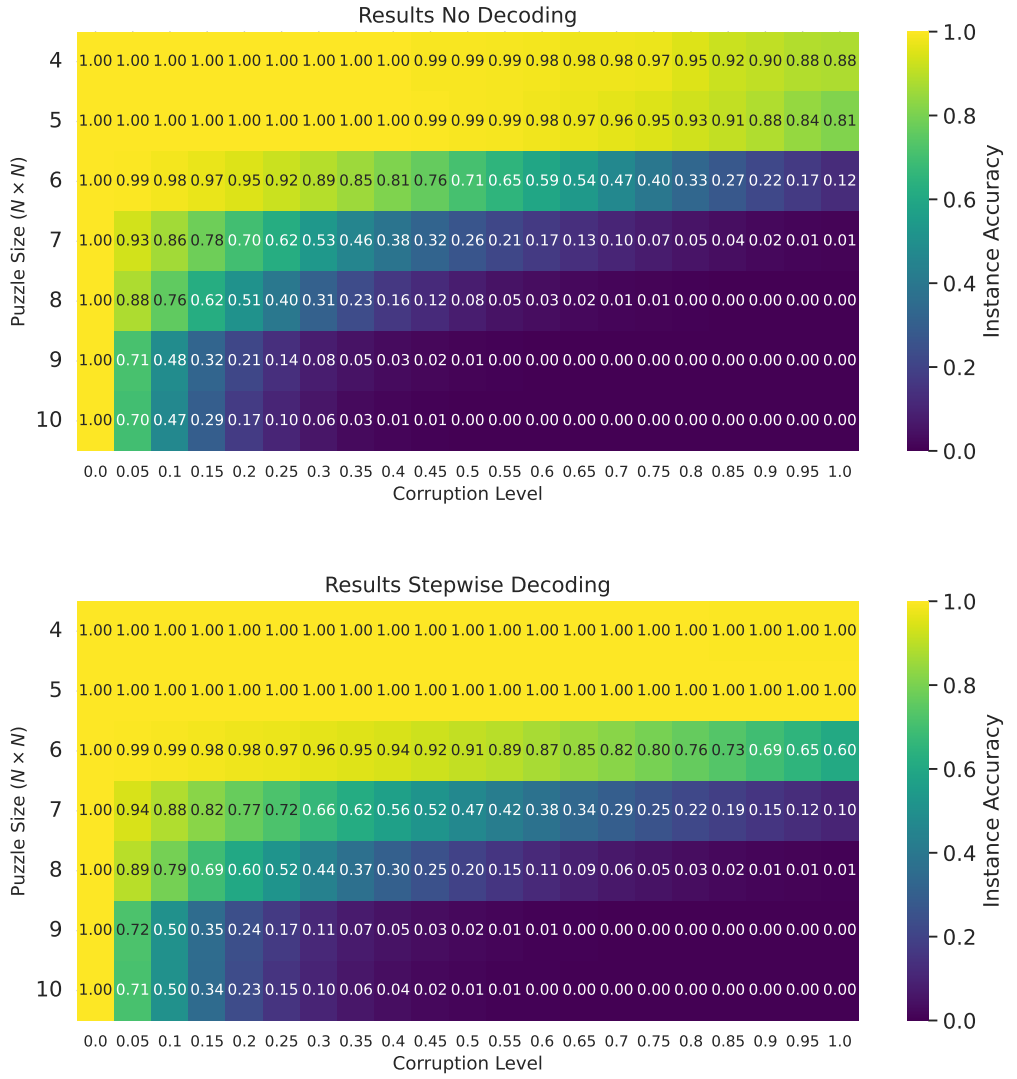

Results Ensemble Decoding
Puzzle Size ($N \times N$)
5 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00
6 1.00 0.98 0.95 0.93 0.91 0.89 0.86 0.84 0.81 0.78 0.75 0.72 0.69 0.66 0.63 0.59 0.56 0.52 0.48 0.44 0.40
7 1.00 0.94 0.90 0.85 0.80 0.75 0.70 0.65 0.61 0.56 0.51 0.47 0.43 0.38 0.34 0.30 0.26 0.23 0.20 0.17 0.14
8 1.00 0.92 0.84 0.76 0.69 0.62 0.55 0.49 0.43 0.37 0.32 0.28 0.23 0.19 0.16 0.13 0.10 0.08 0.06 0.05 0.03
9 1.00 0.84 0.69 0.58 0.47 0.39 0.32 0.26 0.21 0.17 0.13 0.10 0.08 0.06 0.04 0.03 0.02 0.02 0.01 0.01 0.01
10 1.00 0.80 0.64 0.51 0.40 0.32 0.24 0.18 0.14 0.10 0.07 0.05 0.04 0.03 0.02 0.01 0.01 0.00 0.00 0.00 0.00
0.0 0.05 0.1 0.15 0.2 0.25 0.3 0.35 0.4 0.45 0.5 0.55 0.6 0.65 0.7 0.75 0.8 0.85 0.9 0.95 1.0
Corruption Level
Instance Accuracy
1.0
0.8
0.6
0.4
0.2

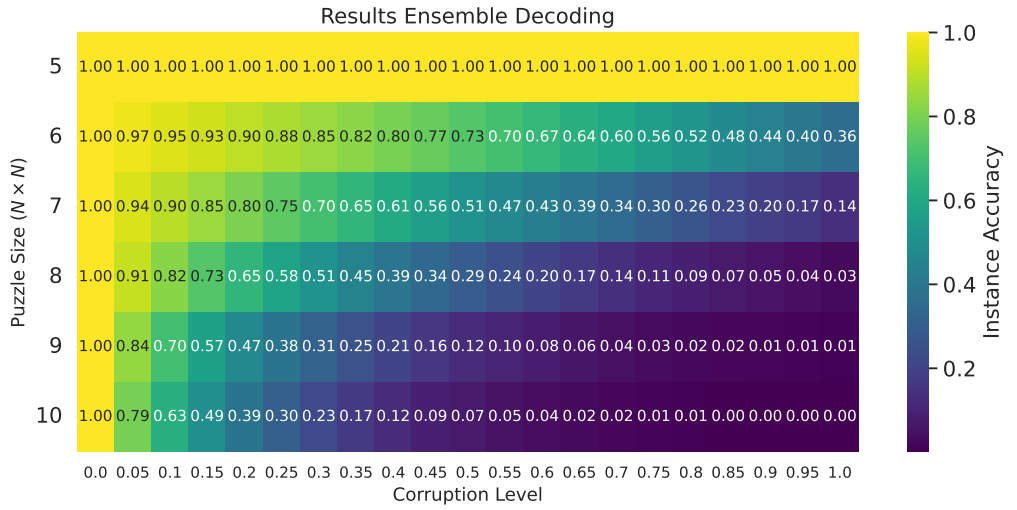

Performance Gain Stepwise Decoding vs. No Decoding
Puzzle Size ($N \times N$)
4 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.01 0.02 0.02 0.02 0.02 0.02 0.03 0.03 0.06 0.07 0.08 0.09 0.10
5 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.01 0.01 0.02 0.02 0.03 0.05 0.07 0.09
6 0.00 0.00 0.01 0.02 0.03 0.05 0.07 0.10 0.13 0.16 0.19 0.22 0.25 0.29 0.32 0.35 0.38 0.40 0.42 0.42 0.42
7 0.00 0.01 0.02 0.04 0.07 0.10 0.13 0.16 0.19 0.20 0.21 0.21 0.21 0.19 0.18 0.16 0.15 0.13 0.11 0.09 0.07
8 0.00 0.01 0.03 0.06 0.08 0.10 0.11 0.12 0.12 0.11 0.10 0.08 0.07 0.05 0.04 0.03 0.02 0.01 0.01 0.01 0.00
9 0.00 0.01 0.01 0.02 0.02 0.02 0.02 0.02 0.01 0.01 0.01 0.01 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
10 0.00 0.01 0.02 0.04 0.04 0.04 0.03 0.02 0.02 0.01 0.01 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.0 0.05 0.1 0.15 0.2 0.25 0.3 0.35 0.4 0.45 0.5 0.55 0.6 0.65 0.7 0.75 0.8 0.85 0.9 0.95 1.0
Corruption Level
Absolute Instance Accuracy Gain
0.5
0.4
0.3
0.2
0.1
0.0
−0.1

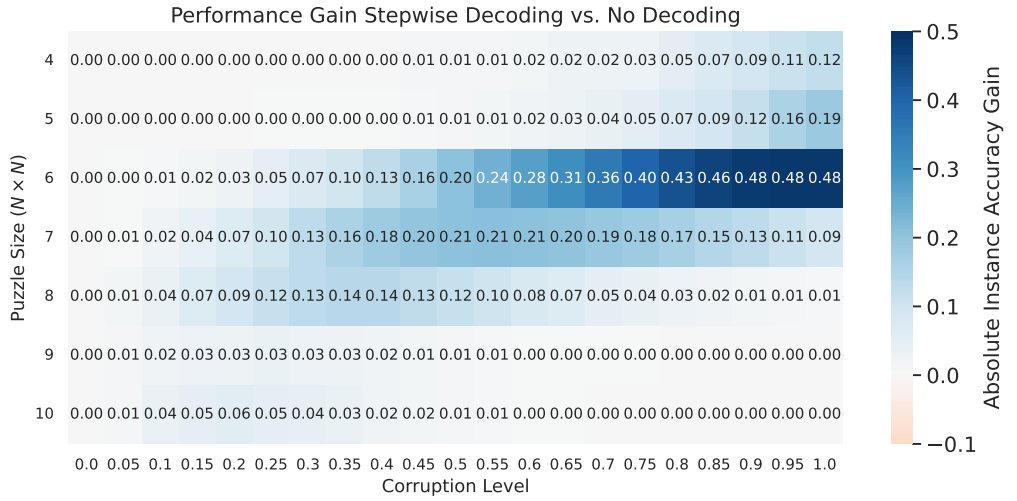

Performance Gain Ensemble Decoding vs. Stepwise Decoding
Puzzle Size ($N \times N$)
5 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
6 0.00 -0.02 -0.04 -0.05 -0.07 -0.08 -0.10 -0.11 -0.12 -0.13 -0.15 -0.16 -0.16 -0.17 -0.17 -0.18 -0.19 -0.19 -0.19 -0.18 -0.17
7 0.00 0.01 0.01 0.03 0.03 0.03 0.04 0.04 0.05 0.05 0.05 0.06 0.07 0.07 0.07 0.08 0.07 0.07 0.07 0.07 0.06
8 0.00 0.04 0.06 0.09 0.11 0.13 0.14 0.14 0.15 0.15 0.15 0.14 0.13 0.12 0.10 0.09 0.08 0.06 0.05 0.04 0.03
9 0.00 0.11 0.18 0.22 0.22 0.21 0.20 0.18 0.16 0.13 0.11 0.09 0.07 0.06 0.04 0.03 0.02 0.02 0.01 0.01 0.01
10 0.00 0.09 0.15 0.17 0.18 0.17 0.15 0.13 0.10 0.08 0.06 0.05 0.03 0.03 0.02 0.01 0.01 0.00 0.00 0.00 0.00
0.0 0.05 0.1 0.15 0.2 0.25 0.3 0.35 0.4 0.45 0.5 0.55 0.6 0.65 0.7 0.75 0.8 0.85 0.9 0.95 1.0
Corruption Level
Absolute Instance Accuracy Gain
0.5
0.4
0.3
0.2
0.1
0.0
−0.1

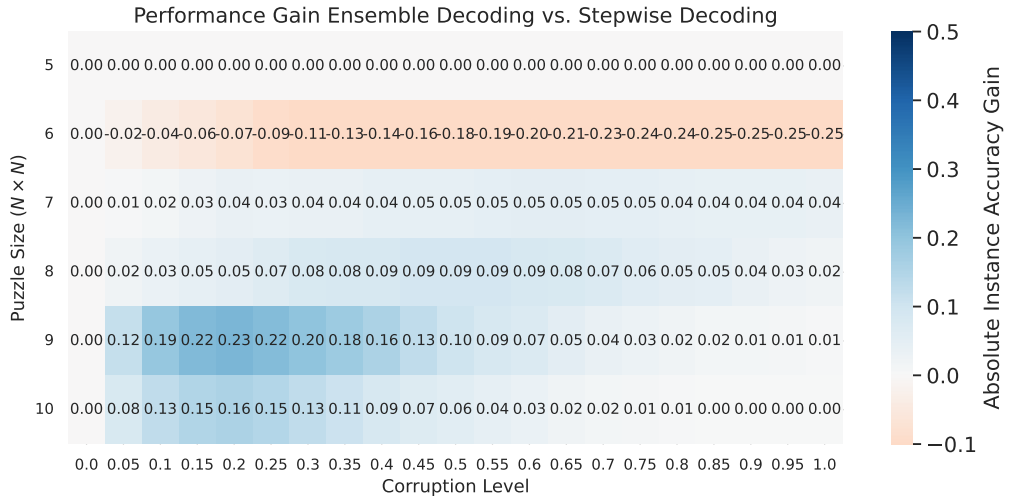
A.4 Random RoPE Cutting

As mentioned previously. During training we take random cuts of RoPE encoding from 10x10 grid and apply it to the 5x5 grid. To ablate whether this needed, we train a separate model with the same objective as previously, just leaving only 5x5 grid RoPE as positional encodings for the token and we run the same suite of test as before. We present the equivalents of the Figures discussed above for this model below. We do not provide commentary as they are just analogues of the previously stated figures. We observe a decrease of performance across sizes. This is as expected, as the model is now not used to seeing different encodings, and therefore cannot perform as well outside of distribution.




Results Ensemble Decoding
Puzzle Size ($N \times N$)
5 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00
6 1.00 0.97 0.95 0.93 0.90 0.88 0.85 0.82 0.80 0.77 0.73 0.70 0.67 0.64 0.60 0.56 0.52 0.48 0.44 0.40 0.36
7 1.00 0.94 0.90 0.85 0.80 0.75 0.70 0.65 0.61 0.56 0.51 0.47 0.43 0.39 0.34 0.30 0.26 0.23 0.20 0.17 0.14
8 1.00 0.91 0.82 0.73 0.65 0.58 0.51 0.45 0.39 0.34 0.29 0.24 0.20 0.17 0.14 0.11 0.09 0.07 0.05 0.04 0.03
9 1.00 0.84 0.70 0.57 0.47 0.38 0.31 0.25 0.21 0.16 0.12 0.10 0.08 0.06 0.04 0.03 0.02 0.02 0.01 0.01 0.01
10 1.00 0.79 0.63 0.49 0.39 0.30 0.23 0.17 0.12 0.09 0.07 0.05 0.04 0.02 0.02 0.01 0.01 0.00 0.00 0.00 0.00
0.0 0.05 0.1 0.15 0.2 0.25 0.3 0.35 0.4 0.45 0.5 0.55 0.6 0.65 0.7 0.75 0.8 0.85 0.9 0.95 1.0
Corruption Level
Instance Accuracy
1.0
0.8
0.6
0.4
0.2


Performance Gain Stepwise Decoding vs. No Decoding
Puzzle Size ($N \times N$)
4 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.01 0.01 0.01 0.02 0.02 0.02 0.03 0.05 0.07 0.09 0.11 0.12
5 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.01 0.01 0.01 0.02 0.03 0.04 0.05 0.07 0.09 0.12 0.16 0.19
6 0.00 0.00 0.01 0.02 0.03 0.05 0.07 0.10 0.13 0.16 0.20 0.24 0.28 0.31 0.36 0.40 0.43 0.46 0.48 0.48 0.48
7 0.00 0.01 0.02 0.04 0.07 0.10 0.13 0.16 0.18 0.20 0.21 0.21 0.21 0.20 0.19 0.18 0.17 0.15 0.13 0.11 0.09
8 0.00 0.01 0.04 0.07 0.09 0.12 0.13 0.14 0.14 0.13 0.12 0.10 0.08 0.07 0.05 0.04 0.03 0.02 0.01 0.01 0.01
9 0.00 0.01 0.02 0.03 0.03 0.03 0.03 0.03 0.02 0.01 0.01 0.01 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
10 0.00 0.01 0.04 0.05 0.06 0.05 0.04 0.03 0.02 0.02 0.01 0.01 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.0 0.05 0.1 0.15 0.2 0.25 0.3 0.35 0.4 0.45 0.5 0.55 0.6 0.65 0.7 0.75 0.8 0.85 0.9 0.95 1.0
Corruption Level
Absolute Instance Accuracy Gain
0.5
0.4
0.3
0.2
0.1
0.0
−0.1


Performance Gain Ensemble Decoding vs. Stepwise Decoding
Puzzle Size ($N \times N$)
5 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
6 0.00 -0.02 -0.04 -0.06 -0.07 -0.09 -0.11 -0.13 -0.14 -0.16 -0.18 -0.19 -0.20 -0.21 -0.23 -0.24 -0.24 -0.25 -0.25 -0.25 -0.25
7 0.00 0.01 0.02 0.03 0.04 0.03 0.04 0.04 0.04 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.04 0.04 0.04 0.04 0.04
8 0.00 0.02 0.03 0.05 0.05 0.07 0.08 0.08 0.09 0.09 0.09 0.09 0.09 0.08 0.07 0.06 0.05 0.05 0.04 0.03 0.02
9 0.00 0.12 0.19 0.22 0.23 0.22 0.20 0.18 0.16 0.13 0.10 0.09 0.07 0.05 0.04 0.03 0.02 0.02 0.01 0.01 0.01
10 0.00 0.08 0.13 0.15 0.16 0.15 0.13 0.11 0.09 0.07 0.06 0.04 0.03 0.02 0.02 0.01 0.01 0.00 0.00 0.00 0.00
0.0 0.05 0.1 0.15 0.2 0.25 0.3 0.35 0.4 0.45 0.5 0.55 0.6 0.65 0.7 0.75 0.8 0.85 0.9 0.95 1.0
Corruption Level
Absolute Instance Accuracy Gain
0.5
0.4
0.3
0.2
0.1
0.0
−0.1

Bibliography

[1] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, "Finetuned language models are zero-shot learners," *arXiv preprint arXiv:2109.01652*, 2021.

[2] B. Estermann and R. Wattenhofer, "Reasoning effort and problem complexity: A scaling analysis in llms," *arXiv preprint arXiv:2503.15113*, 2025.

[3] P. Veličković and C. Blundell, "Neural algorithmic reasoning," *Patterns*, vol. 2, no. 7, 2021.

[4] P. Veličković, A. P. Badia, D. Budden, R. Pascanu, A. Banino, M. Dashevskiy, R. Hadsell, and C. Blundell, "The clrs algorithmic reasoning benchmark," in *International Conference on Machine Learning*. PMLR, 2022, pp. 22 084–22 102.

[5] B. Estermann, L. A. Lanzendörfer, Y. Niedermayr, and R. Wattenhofer, "Puzzles: A benchmark for neural algorithmic reasoning," *Advances in Neural Information Processing Systems*, vol. 37, pp. 127 059–127 098, 2024.

[6] N. Grillo, A. Toccaceli, J. Mathys, B. Estermann, S. Fresca, and R. Wattenhofer, "Beyond interpolation: Extrapolative reasoning with reinforcement learning and graph neural networks," *arXiv preprint arXiv:2502.04402*, 2025.

[7] P. Veličković, R. Ying, M. Padovano, R. Hadsell, and C. Blundell, "Neural execution of graph algorithms," *arXiv preprint arXiv:1910.10593*, 2019.

[8] S. Mahdavi, K. Swersky, T. Kipf, M. Hashemi, C. Thrampoulidis, and R. Liao, "Towards better out-of-distribution generalization of neural algorithmic reasoning tasks," *arXiv preprint arXiv:2211.00692*, 2022.

[9] B. Y. Lin, R. L. Bras, K. Richardson, A. Sabharwal, R. Poovendran, P. Clark, and Y. Choi, "Zebralogic: On the scaling limits of llms for logical reasoning," *arXiv preprint arXiv:2502.01100*, 2025.

[10] J. Chen, Q. He, S. Yuan, A. Chen, Z. Cai, W. Dai, H. Yu, Q. Yu, X. Li, J. Chen *et al.*, "Enigmata: Scaling logical reasoning in large language models with synthetic verifiable puzzles," *arXiv preprint arXiv:2505.19914*, 2025.

[11] P. Giannoulis, Y. Pantis, and C. Tzamos, "Teaching transformers to solve combinatorial problems through efficient trial & error," *arXiv preprint arXiv:2509.22023*, 2025.

[12] J. Schultz, J. Adamek, M. Jusup, M. Lanctot, M. Kaisers, S. Perrin, D. Hennes, J. Shar, C. Lewis, A. Ruoss *et al.*, "Mastering board games by external and internal planning with language models," *arXiv preprint arXiv:2412.12119*, 2024.

[13] Y. Du, J. Mao, and J. B. Tenenbaum, "Learning iterative reasoning through energy diffusion," *arXiv preprint arXiv:2406.11179*, 2024.

[14] A. Oarga and Y. Du, "Generalizable reasoning through compositional energy minimization," *arXiv preprint arXiv:2510.20607*, 2025.

[15] Y. Li, F. Gimeno, P. Kohli, and O. Vinyals, "Strong generalization and efficiency in neural programs," *arXiv preprint arXiv:2007.03629*, 2020.

[16] G. Rodionov and L. Prokhorenkova, "Discrete neural algorithmic reasoning," *arXiv preprint arXiv:2402.11628*, 2024.

[17] L. Saldyt and S. Kambhampati, "Mind the gap: Deep learning doesn't learn deeply," *arXiv preprint arXiv:2505.18623*, 2025.

[18] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020.

[19] C. Wewer, B. Pogodzinski, B. Schiele, and J. E. Lenssen, "Spatial reasoning with denoising models," *arXiv preprint arXiv:2502.21075*, 2025.

[20] J. Austin, D. D. Johnson, J. Ho, D. Tarlow, and R. Van Den Berg, "Structured denoising diffusion models in discrete state-spaces," *Advances in neural information processing systems*, vol. 34, pp. 17 981–17 993, 2021.

[21] S. Nie, F. Zhu, Z. You, X. Zhang, J. Ou, J. Hu, J. Zhou, Y. Lin, J.-R. Wen, and C. Li, "Large language diffusion models," *arXiv preprint arXiv:2502.09992*, 2025.

[22] J. Ye, J. Gao, S. Gong, L. Zheng, X. Jiang, Z. Li, and L. Kong, "Beyond autoregression: Discrete diffusion for complex reasoning and planning," *arXiv preprint arXiv:2410.14157*, 2024.

[23] B. Chamberlain, J. Rowbottom, M. I. Gorinova, M. Bronstein, S. Webb, and E. Rossi, "Grand: Graph neural diffusion," in *International conference on machine learning*. PMLR, 2021, pp. 1407–1418.

[24] Z. Sun and Y. Yang, "Difusco: Graph-based diffusion solvers for combinatorial optimization," *Advances in neural information processing systems*, vol. 36, pp. 3706–3731, 2023.

[25] H. Zhao, K. Yu, Y. Huang, R. Yi, C. Zhu, and K. Xu, "Disco: Efficient diffusion solver for large-scale combinatorial optimization problems," *arXiv preprint arXiv:2406.19705*, 2024.

[26] Z.-H. Fu, K.-B. Qiu, and H. Zha, "Generalize a small pre-trained model to arbitrarily large tsp instances," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 8, 2021, pp. 7474–7482.

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

[28] J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu, "Roformer: Enhanced transformer with rotary position embedding," *Neurocomputing*, vol. 568, p. 127063, 2024.

[29] E. Hoogeboom, D. Nielsen, P. Jaini, P. Forré, and M. Welling, "Argmax flows and multinomial diffusion: Learning categorical distributions," *Advances in neural information processing systems*, vol. 34, pp. 12 454–12 465, 2021.

[30] S. Sahoo, M. Arriola, Y. Schiff, A. Gokaslan, E. Marroquin, J. Chiu, A. Rush, and V. Kuleshov, "Simple and effective masked diffusion language models," *Advances in Neural Information Processing Systems*, vol. 37, pp. 130 136–130 184, 2024.

[31] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[32] L. N. Smith and N. Topin, "Super-convergence: Very fast training of neural networks using large learning rates," in *Artificial intelligence and machine learning for multi-domain operations applications*, vol. 11006. SPIE, 2019, pp. 369–386.

[33] C. R. N. Tassi, J. Gawlikowski, A. U. Fitri, and R. Triebel, "The impact of averaging logits over probabilities on ensembles of neural networks." in *AISafety@ IJCAI*, 2022, pp. 132–141.

[34] R. Razafindralambo, R. Sun, F. Precioso, D. Garreau, and P.-A. Mattei, "When are two scores better than one? investigating ensembles of diffusion models," *arXiv preprint arXiv:2601.11444*, 2026.