σ-GPTs: A New Approach to Autoregressive Models

Author information scrubbed for double-blind reviewing

No Institute Given

Abstract. Autoregressive models, such as the GPT family, use a fixed order, usually left-to-right, to generate sequences. However, this is not a necessity. In this paper, we challenge this assumption and show that by simply adding a positional encoding for the output, this order can be modulated on-the-fly per-sample which offers key advantageous properties. It allows for the sampling of and conditioning on arbitrary subsets of tokens, and it also allows sampling in one shot multiple tokens dynamically according to a rejection strategy, leading to a sub-linear number of model evaluations. We evaluate our method across various domains, including language modeling, path-solving, and aircraft vertical rate prediction, decreasing the number of steps required for generation by an order of magnitude.

Keywords: Autoregressive models · Permutations · Transformers · Rejection Sampling

1 Introduction

Transformers demonstrate exceptional autoregressive capabilities across modalities. The traditional take for autoregression is to follow the natural order of the data, for example, left-to-right for text. In the case of vision, the usual scheme is to unfold the images following a raster-scan order and to use transformers to model the obtained sequence. In this work, we make a distinction between the order of the input data and the order of autoregression, highlighting that while they are typically aligned in most applications, they need not be. Our investigation involves training and generating sequences in a randomly shuffled order using transformers. While changing the sequence order is more challenging during training, it also reveals fascinating properties of the models.

By breaking away from the standard autoregression order, one can use the model to predict the tokens in any particular order. With this scheme, the model is capable of predicting at any moment of the generation the conditional distribution of the remaining tokens. Having these estimates allows quantifying the possible outcomes of the generation at any given point. More interestingly, they can be leveraged to do rejection sampling, allowing to generate sequences by burst with a dynamical number of steps.

This work is structured as follows, we first introduce σ-GPTs and shuffled autoregression, and show that a model trained with this method combined with
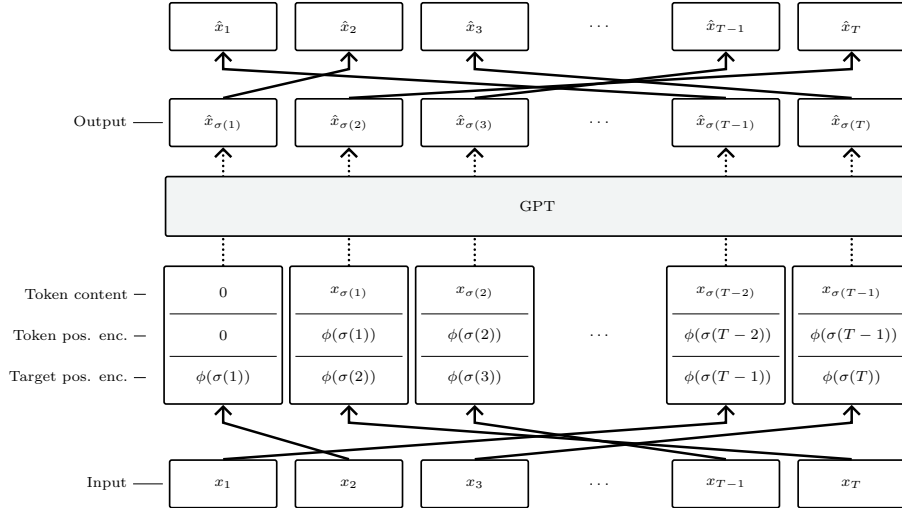
Fig. 1: In our σ-GPT, an arbitrary shuffling order σ can be chosen on-the-fly for every sample. It induces an input order $0, \sigma(1), \sigma(2), \ldots$ and an output order $\sigma(1), \sigma(2), \sigma(3), \ldots$, where the input is first padded with a 0 to ensure a consistent number of tokens. Tokens are shuffled accordingly, and these orders are both encoded separately with two positional encodings concatenated to the input, allowing the model to sample consistently in the autoregressive process. The output is finally shuffled back to the true order.

	σ-**GPT**	**GPT**	**Diffusion Models**
Sample Anywhere	✔	✗	✗
Conditional Density Estimation	✔	✗	✗
Arbitrary conditioning	✔	✔	∼
Infilling	✔	✗	∼
Burst Sampling	✔	✗	✔
Log-likelihood Training	✔	✔	✗

Table 1: Comparison between our approach, a standard causal transformer encoder (called GPT here), and diffusion models. Our model allows the sampling of a token at any position in the sequence, to model the remaining density according to a partially sampled sequence, naturally supports infilling, and can be used to sample the sequence by burst allowing faster generation. Compared to diffusion models, it can be trained easily using cross-entropy.

a curriculum method can even increase the performance of the underlying model. We then present the additional properties of σ-GPTs, summarized in Table 1, in particular for estimating conditional probabilities and we present our token-based rejection sampling scheme which allows for generating the sequence per burst and its theoretical properties. We evaluate our model and our scheme on three main tasks, which are open text generation, path-solving, and aircraft vertical rate prediction.

Contributions

- Introduce σ-GPT, a novel architecture, with two positional encodings related respectively to the input and output order, that allows a causal transformer to generate sequences in any order which can be modulated on the fly for any pass through the model.
- Demonstrate that our method can reach similar performance as left-to-right trained autoregressive models when trained with a curriculum scheme.
- Demonstrate that our method can be used to generate samples in any order, allowing for the generation of samples conditioned on any part of the sequence.
- Introduce a novel token-based rejection sampling scheme that leads to the generation of samples per burst.

2   Methodology

2.1   σ-GPTs: Shuffled Autoregression

We propose a novel approach for training autoregressive models, which involves doing next-token prediction on a shuffled input sequence. We present σ-GPT, where σ denotes the permutation used to shuffle the sequence, and by GPT we mean any causal transformer encoder (or causal transformer decoder without cross-attention) such as [12]. To train such a model, each sequence is shuffled randomly during training. The model is then tasked to predict the next token in the shuffled sequence conditioned on all the tokens it has seen before. This training is done as usual with a standard cross-entropy loss. Besides the randomization of the order of the sequence and the addition of a double positional encoding, no other changes are needed to the model or training pipelines. For the rest of the paper, we use 'left-to-right order' to mention the usual order in which models are trained, even in the case of 2D data which are usually mapped to a sequence using a raster-scan order. And we use 'random order' to mean that the input has been shuffled.

2.2   Double Positional Encodings

To be able to model sequences in any order, each token needs to have information about its position and the one of the next token in the shuffled sequence. Specifically, when handling a sequence of tokens alongside a given permutation

σ, every token contains three distinct pieces of information: its value $x_{\sigma(t)}$, its current position $\sigma(t)$, and the position $\sigma(t + 1)$ of the subsequent token in the shuffled sequence, that are all concatenated. The necessity for double positional encoding arises from the intrinsic characteristics of transformers. Given that each token attends to every previous token in a position-invariant manner, each token needs to contain information about its position in the original sequence, so other tokens can know where they are located. And each token needs to know the position of the next token in the shuffled sequence as it is the target of the prediction. The double positional encoding is the only architectural change needed to train autoregressive models in random order. In this work, we used the standard sinusoidal positional encoding [14] for both the input and output positional encodings.

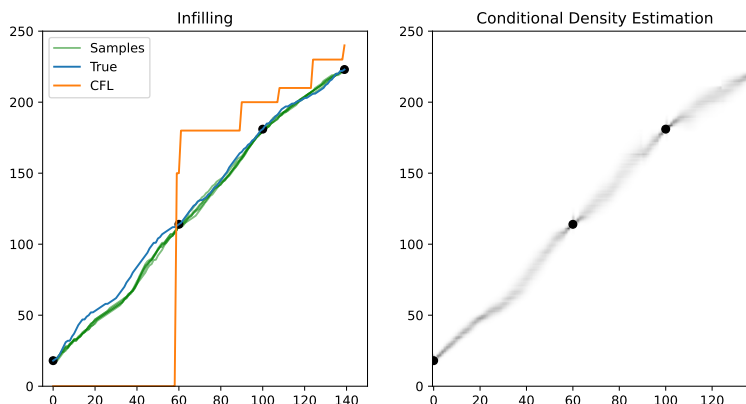## 2.3   Conditional Probabilities and Infilling

Fig. 2: (Left.) We can infill the sequence by conditioning on the known part (black points). (Right.) We can also have estimates of the density at any point of the sequence.

Our method allows making conditional density estimation of the rest of the sequence. It is capable of making predictions all over the task space conditioned on any known subpart of the task. This can be done by prompting the model with the known part of the sequence and then decoding, in parallel and in one pass, the remaining tokens. Such evaluations are not possible with autoregressive models trained in a left-to-right order, as they need to follow the specific order they've been trained in. Examples, showing that the model usually has good estimates of the unconditioned distribution, can be seen in Figures 2 and 3a.
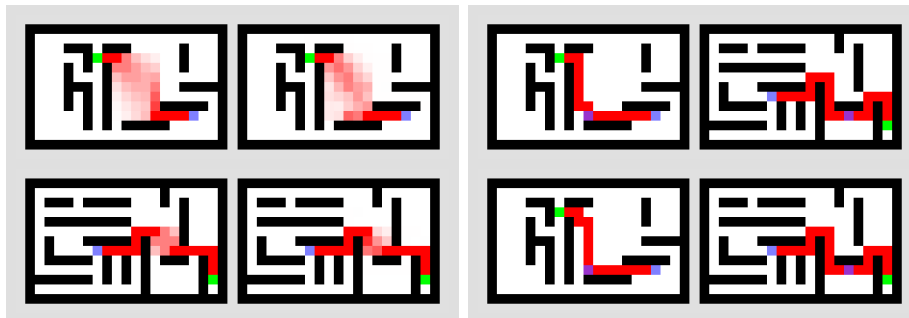
(a) (Left.) The theoretical density of the optimal path in the maze. (Right.) The estimated probability of the class 'path' at every position before starting autoregression. We see that the model has good estimates of the true density.

(b) Two different conditional samplings for each maze. The known part of the path (purple) is prompted first, and the rest of the sequence can be completed coherently.

Fig. 3: Conditional density estimation and infilling on the maze path-solving task.

Directly related to conditional density estimation, is that our method naturally supports infilling, as it is straightforward to prompt the model with the known part of a signal and to decode auto-regressively or by burst the rest of the signal. Figures 2 and 3a shows example of such samplings.

2.4 Token-based Rejection Sampling

Autoregressive generation is a slow process as each token has to be generated sequentially. Even with caching strategies, this still scales linearly with the sequence length and it becomes prohibitively expensive for long sequences [15]. As our model allows for the generation of tokens in any order, we can leverage that fact and sample tokens in parallel at every position of the sequence. We can then evaluate the candidate sequence under different orders and accept multiple tokens in one pass. This algorithm runs efficiently on GPU as both the sampling at every position and the evaluation under different orders can be made in parallel, in a forward pass, and using an adapted KV-caching mechanism. We describe this caching mechanism more in detail in Section 3 of the supplementary material. When conditioned on partially completed sequences the model outputs distributions that are compatible with different possible outcomes, and when evaluating under different orders for generation, the distribution of tokens is constrained to tokens that are compatible with the previous tokens seen in one given order. As both the sampling and evaluation can be done in parallel, we can compute the acceptance decision efficiently for every token.

This strategy outputs a decision for each remaining token, but the decisions made by models become sometimes nonsensical when two mutually exclusive

---

Algorithm 1 Token-based rejection sampling, following notation of [4]

Given minimum target length T, y trained σ-GPT, and number of orders N_o
Given a prompt $x_i \in \mathbb{X}$ of length t_0 of initial tokens. ($\mathbb{X}$ can be the empty set)
Set $t = t_0$
while $t < T$ **do**
  In parallel, compute distribution conditioned on prompt $p(x_i|\mathbb{X}), \forall i \in t, \dots, T$
  In parallel, sample at every position $\tilde{x}_i \sim p(x_i|\mathbb{X}), \forall i \in t, \dots, T$
  Draw N_o random order σ and in parallel, compute all logits $q(x_i|\mathbb{X}, \tilde{x}_{\sigma_{<i}}), \forall i \in t, \dots, T$
  In parallel sample $T - t$ $r_t \sim U[0,1]$ from a uniform distribution
  In parallel, compute the acceptance decision $a_i = r_i < \min(1, \frac{p(\tilde{x}_i|\mathbb{X})}{q(\tilde{x}_i|\mathbb{X}, \tilde{x}_{\sigma_{<i}})})$ for every order
  Select the order that accepts the most tokens before seeing a first rejection
  Keep that order and add the a accepted tokens before the first rejection to the prompt
  Set $t = t + a$
end while

---

tokens are part of the prompt. Once a rejection is seen, all subsequent accepted tokens in the order of evaluation should be discarded. Indeed, the scheme rejects tokens that are incoherent with the ones already seen, and asking a model to make predictions based on incoherent tokens might lead to incoherent decisions. Using multiple orders allows keeping the one that accepts the most tokens in its evaluation. Even if it is dynamic, this algorithm can still easily generate multiple samples at once, by accepting the same amount of tokens for each sequence in the batch. Our rejection sampling algorithm is given in pseudo-code in Algorithm 1.

Other models such as Mask Git [3] or diffusion models [7,1] are doing generation by burst. However, these models usually require fixing the number of steps or a masking schedule beforehand. Our method on the other hand adapts dynamically to the underlying statistics of the data and thus does not require this extra hyper-parameter. We evaluate it on three synthetic cases to showcase this dynamic capability, we present the results in Section 3.8.

2.5   Other orders

Our double positional encoding scheme allows for training and evaluating models in any order. Using a randomized order during training allows conditional density estimation, infilling, and burst sampling at inference time. However the double positional encoding scheme allows any order to be used, and it can be used to train models in a deterministic order that is not left-to-right. As an example, we use a deterministic 'fractal' order to see how it compares to a random or left-to-right order. This order starts in the middle of the sequence then recursively goes to the first quarter and three-quarters of the sequence, and goes on recursively until all the positions have been visited. Such an order is fully deterministic, yet we make the hypothesis that this order leads to more difficult training for

the model as it cannot rely on the locality of the information. We present the results in Section 3.5. Note that under perfect models, the order of modeling and decoding should not matter because of the chain rules of probability. We give more details about it in Section 1.1 of the supplementary material.

2.6 Denoising diffusion models

Denoising diffusion models [7] is a family of generative models that can also be used to generate sequences in a few steps. They are trained to reverse a diffusion process that is applied to the data. Diffusion processes can be both continuous and discrete. In this work, we use as a baseline only the discrete diffusion case, in particular using a uniform diffusion process [1]. To be able to compare the methods fairly, we use the same transformer architecture for both σ-GPT and the diffusion model, changing only the training objective. Compared to σ-GPT, diffusion models are not dynamic and require a fixed number of steps to generate a sequence, independently of the underlying statistics of the data. They also don't natively support conditional density estimation and infilling.

3 Results

3.1 General performance

We tested our model across three main distinct tasks: language modeling, maze path solving, and aircraft vertical-rate prediction.

- Language Modeling: We used both the GPT-2 (123M) model on the Wikitext-103 dataset [10] and GPT-2 (345M) on OpenWeb Text [5].
- Maze Path Solving: This task involves determining a valid path between a starting and ending point in 13 x 21 mazes featuring 15 barriers. Presented with an image of an empty maze with start and end points, the model is tasked with producing an image with a legitimate path.
- Aircraft Vertical-Rate Prediction: This task uses real aircraft trajectory data, with its aircraft type. The data represents trajectories conditioned by air traffic control directives. The model's objective is to predict the vertical trajectory from a plane's current altitude to a specified control level.

Additionally to these tasks, we created a synthetic benchmark for evaluating our burst sampling algorithm.

- Product Dataset: This toy example represents a pure product law case and is made of a sequence of length 100 with two classes (0,1) given by a Bernoulli law with p = 10%.
- Step Dataset: This toy example comprises sequences of two classes (0,1) of length 100 which are 0 everywhere except on a step of length 10 placed randomly in the sequence

- Joint Law dataset: This toy example represents a pure joint law and consists of a sequence of length 100 with 100 different classes, the model should predict a random generation of these different classes.

The general results of our models are presented in Table 2. These results indicate that training in a random order while requiring more compute-time as we describe in Section 3.2, reaches similar performances to left-to-right trained models. For the text modeling, to have a fair comparison during training, we monitor the validation perplexity of the sequence evaluated in a left-to-right order. Training in random order for text modeling was plateauing at a higher left-to-right validation perplexity, but using a curriculum scheme allows reaching the same performances, as presented in Section 3.3. For the path solving and the vertical rate prediction, the models were able to reach the same left-to-right validation loss during training. In inference, we noticed a one percent drop in accuracy compared to diffusion models and left-to-right trained GPT. For the vertical rate prediction task, the dataset that we used is limited to around 23.000 different sequences, we noticed that the standard left-to-right GPT was sometimes stuck repeating the same altitude, we think this is a modeling issue due to the small data regime. σ-GPT does not seem to suffer as much from this problem and offers a decrease in MSE. We hypothesize that this behavior comes from using a random order in inference which forces the model to fix some tokens over the whole sequence early in the generation. By doing so, the model gains the advantage of having a sketch of the whole sample and then concentrates on completing a coherent sample.

Table 2: General results. We report the validation perplexity for text generation, the test accuracy for the maze solver, and the mean squared error (MSE) for the vertical rate prediction. σ-GPT reaches a similar performance as GPT in text generation and maze solving and it outperforms GPT in the case of the vertical rate prediction. We report the validation perplexity for the text generation. For the path solver, we report the test accuracy on 1000 novel mazes. For the vertical prediction task, we report the mean squared error on the test set. We report the mean and standard deviation for the path-solving and the vertical rate prediction task. We do not report the validation error for the text generation for the discrete diffusion (Dis. Diff) as the training objective is different.

	Text-generation		Path Solving	Vertical Rate
	OWT Val Perp. ($\downarrow$)	Wiki-103 Val Perp. ($\downarrow$)	Accuracy ($\uparrow$)	MSE ($\downarrow$)
GPT	18.14	20.30	99.60 ± 0.70	274.8 ± 70.7
σ-GPT	18.64	16.69	98.30 ± 0.67	141.4 ± 4.1
Dis. Diff.	-	-	99.20 ± 0.67	105.94 ± 1.3

Table 3: Training efficiency. Number of steps/epochs required to reach the same performance and comparison with as GPT trained causally. As learning to predict in any order is a more challenging task, it is expected to need more computing time to reach the same accuracy. We don't report the standard deviation for text generation as we limited the training to one run.

Order	Text-generation	Maze Solver	Climbing Rate
σ-**GPT**	32500	78.0 ± 6.5	110.7 ± 4.5
GPT	16500	19.3 ± 4.9	25.0 ± 3.6

3.2   Training Efficiency

Modeling sequences in a random order is a more challenging task than modeling in left-to-right order. We think this is due to two main factors, at the beginning of the sequences models cannot rely on adjacent tokens to make educated guesses for the next token. Second some tasks are harder to learn in one direction than another and by modeling the data in any direction, we are always in the harder scenario. We give an example of one task that is harder to learn in one direction in Section 1 of the supplementary material.

This implies that we expect and see an increase in the number of steps or epochs required to learn a task. As previously mentioned, we don't see experimentally a drop in the validation performance of our model in the case of the path-finding algorithm or the vertical rate forecasting, but the time to reach the same performance increased. In the case of text modeling, the models plateaued before reaching the same accuracy when trained in random order. We treat that case in the following section. We report in table Table 3, the increase in training steps or epoch to reach the same accuracy. We see that most of the time, the number of epochs or steps needed to reach the same performance drastically increases. We think again that this is due to the increased complexity of modeling the sequence without having to rely on local information.

3.3   Curriculum Learning

For text modeling, we found a gap in validation perplexity in the left-to-right order between models trained purely in a random order and models trained in a left-to-right order. We see in Table 4 that σ-GPT is stuck at larger perplexity in both Open Web Text and WikiText-103 (30.43 vs 18.14 and 39.85 vs 20.30). We found that training for longer and using larger model didn't help in reducing that gap. To solve that problem, we introduced a curriculum learning scheme where the model is shown first more sequences in left-to-right order and progressively learns to model the sequence randomly. Surprisingly using this scheme helped drastically the model which managed to get even better performance than left-to-right trained transformers in the Wikitext-103 case and reduce drastically the gap for models trained on OpenWebText.

Table 4: Curriculum learning. We monitor the Validation Perplexity using a left-to-right order during training to have a comparable evaluation. We see that there is a gap between the model trained purely in a left-to-right fashion (GPT) and others trained in a random order (σ-GPT). Training for longer and larger models didn't help in removing that gap. We introduce a curriculum learning scheme that starts presenting the model with some percentage of the data (written in the corresponding label) in a left-to-right order at the beginning of the training and goes linearly to 100% of sequence in a random order at the end of the training. We see that training with this scheme removes the gap between σ-GPT and regular GPT and it reaches even better than left-to-right performance in the WikiText-103 case.

	Text-generation Val Perp. ($\downarrow$)
	Min. Left-to-Right
Openweb Text - GPT (345 M)	
GPT	18.14
σ-GPT curr. 50%	18.64
σ-GPT no curr.	30.43
WikiText 103 - GPT (128M)	
σ-GPT curr. 50%	16.69
σ-GPT curr. 100%	19.38
σ-GPT curr. 10%	19.45
GPT	20.30
σ-GPT no curr.	39.85

3.4   Open Text Generation: t-SNE of generated sequence

To get a qualitative sense of the generated text by the different methods, We generate 3000 sequences of 1024 tokens with each method, embed each sequence using an embedding model, and then project the embeddings to 2D using t-SNE. We present the results in Figure 4. We used Open-AI `text-embedding-3-small` [11] to embed the generated sequences into a single 1536 vector embedding. We represent as green embeddings of sequences of the validation set, used as reference. We compute the t-SNE using the whole 15'000 embeddings and then plot each method (blue) and the other considered method (small gray dots). We first see that embeddings of GPT, σ-GPT, σ-GPT with burst sampling, and diffusion are spread over the whole space, showing that the model can generate sequences that are coherent with the validation set.

3.5   Fractal order

We describe here the results that we get when training a GPT using a deterministic, but not left-to-right order. We described the order in Section 2.5. We train a GPT using this specific order for the different tasks and present the results

(a) Rej. Sampling   (b) σ-GPT   (c) GPT   (d) Diffusion
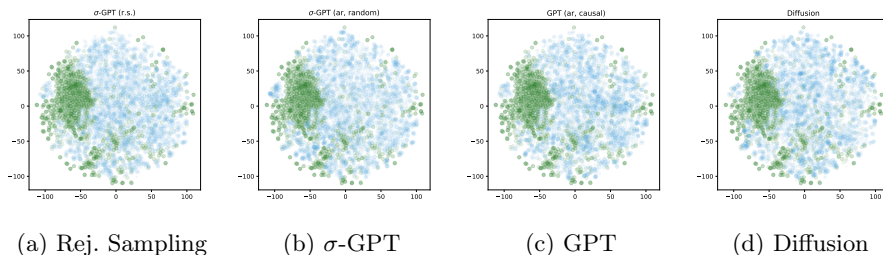
Fig. 4: 2D t-SNE of `text-small-3-embeddings` of 3000 sequences generated by each method. We compute the t-SNE of all the embeddings together, and then we display in each graph the embeddings of the validation set (green), the embeddings of the corresponding method (blue), and the embeddings of the other methods (gray). We see that the embeddings of the generated sequences have the same overall distribution compared to validation sets, which seems to indicate that GPT, σ-GPT, σ-GPT with burst sampling, and diffusion models can generate sequences of similar quality.

in Table 5. We found that training in that order was as difficult for the model as training in a random order, and we noticed a small drop in performance compared to σ-GPT. We suspect that this is due to the high discontinuity of the order of the sequence, which is such that two consecutive tokens are seen far away in the sequences. When predicting the first tokens, the model therefore cannot rely on information contained in neighboring tokens to make its prediction. As the training behavior seen in models trained in random and fractal order is similar, we think that the drop in training efficiency comes more from the fact that the model cannot exploit this neighboring information than changing the order at every batch.

Additionally, models trained in a fractal cannot be used as such for infilling and conditional density estimation and therefore cannot be used with our rejection sampling scheme. As the order is fixed for every batch, it might not even need to have a double positional encoding.

3.6   Memorizing

As learning sequences in any direction is harder than modeling them under a predefined order, we also expect that the critical dataset size when the model switches from memorization to generalization will increase. We follow the same hypotheses than [13], namely that the model has two mechanisms, one generalizing and one memorizing the data. As the mechanism of generalizing is more efficient as the dataset grows it will be selected by gradient descent once the size of the dataset gets beyond a critical size. As learning in a random order is a more difficult task, we expect that generalization is more difficult in that setup as well, hence the memorization regime should hold for bigger dataset sizes. We reduce drastically the training dataset size in the case of the path-finding

	Text-generation Val Perp. (↓), Rand. ord.	**Path Solver** Test Acc (↑)	**Vertical Rate** MSE (↓)
σ-GPT	24.46	98.30 ± 0.67	141.4 ± 4.1
Fractal GPT	27.79	98.00 ± 1.94	145.7 ± 2.6

Table 5: Results for GPT trained in a fractal order compared to a standard left-to-right (GPT) and random (σ-GPT) order. We found that training models in this highly non-continuous order is as hard as training them in a random order, and additionally, models trained in that order cannot be used for conditional density estimation, infilling, or rejection sampling. For text modeling, we report the model perplexity on the validation set, in a random order for σ-GPT and in fractal order for the fractal GPT, as left-to-right validation perplexity is meaningless for fractal GPT which did not see sequences in that order during training.
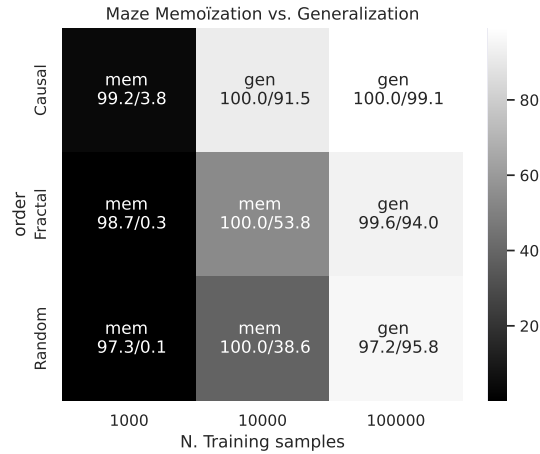

Fig. 5: Number of examples needed to switch from memorization to generalization. The model is trained on a restricted dataset size in the path-finding task. We see that the model trained in a random order needs more examples to switch from memorization to generalization. At 1k samples both models are fully in a memorization regime, at 100k both generalize but in between, at 10k, the model trained in a random order is still in a memorization regime.

task and we present the results in Figure 5. Once it gets to 1000 examples both models trained in left-to-right, fractal, and random order are in a memorizing regime, getting perfect accuracy on the training data but very low on the validation data. Conversely, once the dataset gets bigger than 100k examples models trained in all the different orders are in a generalization regime. The transition happens in between and we find that it happens faster in the left-to-right order: at around 10k samples, the models trained left-to-right can generalize, while models trained in a random order are still in a memorization regime. We see also that models trained in a fractal order start generalizing faster than models trained in a random order, suggesting that the model can rely on seeing always the same order to generalize more rapidly.

3.7   Infilling and Conditional density estimation

We show in Figure 2 that our model can be used to infill the sequence by conditioning on the known part of the sequence. In this figure, the larger points are part of the prompt and one can see the generated sequence complete sequence that matches the prompt. This figure also shows that our model has good estimates of the density at any point of the sequence. We represent at each point in the sequence the probability of the next sample given the known part of the sequence as shades of gray, the darker the more probable. We see that during generation, the model sees multiple possible outcomes that are coherent with the known part of the sequence. They are then constrained to a single sequence during the sampling. For left-to-right trained models, we can only have estimates of the density for the next tokens and we can't know what the model estimates for the rest of the sequence. In the case of the path-finding task, we show in Figure 3a that the model conditioned only on an empty maze has good estimates of the true density of the optimal paths, highlighting that the model has already partially solved the problem before starting generation. During sampling, the order of the generation and the sampling procedure influence which path is selected from this joint law. We show as well in Section 2.3, that we can constrain the generation of mazes and that the model can generate coherent samples based on some prompted tokens that can be chosen on the fly.

We also give some interactive examples of text generation in the supplementary material.

3.8   Token-based Rejection Sampling Scheme

We applied our token-based rejection sampling scheme to the problem of text generation, path-finding, and vertical rate forecasting Figures 6a to 6c. We found that in the three cases, our method was able to generate samples of comparable quality with an order of magnitude fewer steps than the autoregressive method. We compared to discrete diffusion models as well, and we can see that our method always outputs coherent samples, while the samples generated by the diffusion model are sometimes incoherent if the number of steps is not high enough. In the

case of path-solving and in the three synthetic tasks, we see that at a comparable number of steps for both methods, our model shows better generation quality.

We tested our token-based rejection sampling scheme on three synthetic cases. We estimate the number of steps required to generate the sequence using a perfect model in Section 2 of the supplementary material. We found that our scheme was close to the optimal heuristics in all three cases. In the case of the product dataset, requiring only one step to accept the sequence. In the case of the step dataset, our scheme required four steps to accept the sequence. In the case of the joint law dataset, our scheme required a few more steps, which is expected as the task is more complicated. We see that it manages to generate valid samples with a number of steps close to the optimal heuristics and with a large increase in performance compared to diffusion models at the same step.
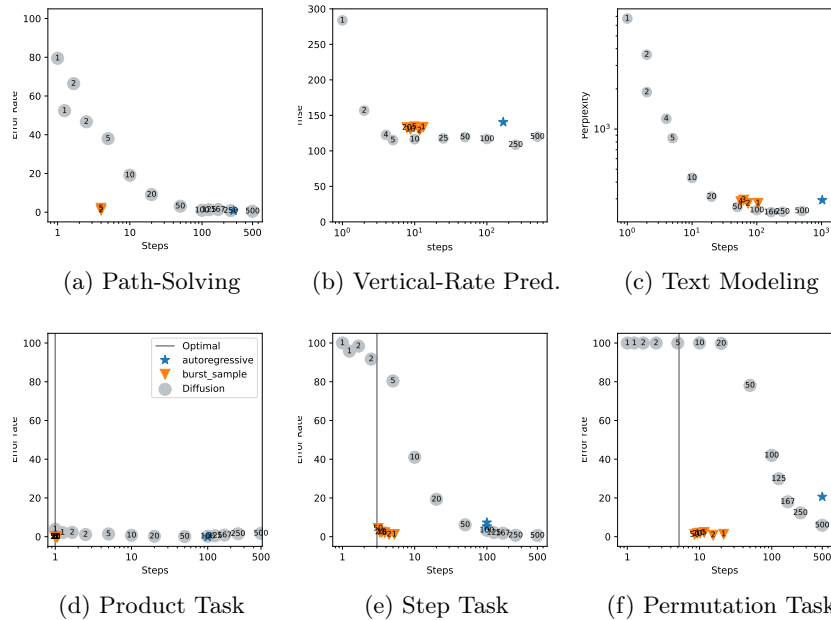

(a) Path-Solving   (b) Vertical-Rate Pred.   (c) Text Modeling

(d) Product Task   (e) Step Task   (f) Permutation Task

Fig. 6: We plot the performance vs steps of our σ-GPT used for autoregression in random order (blue), to σ-GPT with rejection sampling per burst (orange) against diffusion models (gray). We denote in the text the predefined number of steps chosen for generation in the diffusion models. For rejection sampling, we note the number of orders used for the evaluation. We see that increasing the number of orders leads to a decreased number of steps. For the synthetic tasks, we also represent heuristics for the optimal number of steps needed to generate the sequence (gray line), as described in Section 2 of the supplementary material, and we see that our scheme is close to this heuristics.

4   Related works

4.1   Shuffling in language models

In exploring the landscape of shuffling used by autoregressive models, one finds noteworthy precedence in the work of [16]. XLNet, which uses randomization of the sequence order as a pretraining task for sentence encoding in the context of natural language understanding. Our approach is similar with regards to the shuffling of the sequence, and the shuffled language modeling but differs in its implementation: we use double positional encoding and a regular causal mask instead of relying on two streams and the modification of the attention matrix. The objective of the two models differs as well, XLNET is used to encode information similarly to BERT while our approach is generative. A very recent approach by [6], trains a transformer both on a left-to-right and right-to-left order which solves a classic problem of the transformer model to understand the context of the token in both directions. Our approach by default trains the model in any order, and we can consider that it sees an exponential number of different sequences during training. However, our section on memoization shows that this does not serve as a good data augmentation technique and that the model seems to memorize the data more in that case.

4.2   Burst sampling scheme

Other works are trying to solve the problem of the linear time required by autoregression using burst sampling. Maskgit [3], for example, uses a BERT-like masked language model (MLM) and uses a custom decoding scheme, which samples multiple tokens at the same time to generate an output. The number of tokens generated at each pass is fixed by the masking schedule and the model utilizes a confidence-driven decoding scheme to choose which tokens to predict. In a distinct approach, [9] introduced a method that leverages an auxiliary model to guide the generation process. Alternatively, the approach outlined by [8] primarily focuses on generating preliminary drafts of an image, which are then progressively enhanced in subsequent iterations. Following work in video generation [2,15], are leveraging a MaskGit [3] like approach for generation as autoregressive generation of video would be too costly. Our rejection sampling scheme allows as well to generate the sequence by burst but in contrast to another scheme, the number of tokens accepted is dynamic and depends on the data being modelled, which allows faster generation when the underlying data distribution is simple.

5   Conclusion

Training GPT-like models in different orders offer different desirable properties. It allows for the conditional prediction based on any subset of the tokens of the sequence, it naturally can be used for infilling, and as the model can do partial

prediction, we can leverage them to do rejection sampling and accept multiple tokens at the same time during generation. Our findings indicate that conditional prediction learned by the models matches the theoretical partial distribution showing that the model is indeed able to understand and reconstruct the signal in any order. As the training objective of modeling sequences in any order is harder than training in a fixed order, it has an impact on the training efficiency, and in a small dataset size, we show that it leads to more memoïzation. Finally, we showed that our model was able to generate sequences by burst using a novel per-token rejection sampling scheme, reaching optimal heuristics in some cases and decreasing the number of steps needed for generation by an order of magnitude.

References

1. Austin, J., Johnson, D.D., Ho, J., Tarlow, D., Van Den Berg, R.: Structured denoising diffusion models in discrete state-spaces. Advances in Neural Information Processing Systems **34**, 17981–17993 (2021)
2. Chang, H., Zhang, H., Barber, J., Maschinot, A., Lezama, J., Jiang, L., Yang, M.H., Murphy, K., Freeman, W.T., Rubinstein, M., Li, Y., Krishnan, D.: Muse: Text-to-image generation via masked generative transformers (2023)
3. Chang, H., Zhang, H., Jiang, L., Liu, C., Freeman, W.T.: Maskgit: Masked generative image transformer. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 11315–11325 (June 2022)
4. Chen, C., Borgeaud, S., Irving, G., Lespiau, J.B., Sifre, L., Jumper, J.: Accelerating large language model decoding with speculative sampling. arXiv preprint arXiv:2302.01318 (2023)
5. Gokaslan, A., Cohen, V.: Openwebtext corpus. `http://Skylion007.github.io/OpenWebTextCorpus` (2019)
6. Golovneva, O., Allen-Zhu, Z., Weston, J., Sukhbaatar, S.: Reverse training to nurse the reversal curse (2024)
7. Ho, J., Jain, A., Abbeel, P.: Denoising diffusion probabilistic models. Advances in neural information processing systems **33**, 6840–6851 (2020)
8. Lee, D., Kim, C., Kim, S., Cho, M., HAN, W.S.: Draft-and-revise: Effective image generation with contextual rq-transformer. In: Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A. (eds.) Advances in Neural Information Processing Systems. vol. 35, pp. 30127–30138. Curran Associates, Inc. (2022)
9. Lezama, J., Chang, H., Jiang, L., Essa, I.: Improved masked image generation with token-critic. In: Avidan, S., Brostow, G., Cissé, M., Farinella, G.M., Hassner, T. (eds.) Computer Vision – ECCV 2022. pp. 70–86. Springer Nature Switzerland, Cham (2022)
10. Merity, S., Xiong, C., Bradbury, J., Socher, R.: Pointer sentinel mixture models. In: International Conference on Learning Representations (2017)
11. OpenAI: New embedding models and api updates. `https://openai.com/blog/new-embedding-models-and-api-updates` (2024), accessed: [01.03.2024]
12. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al.: Language models are unsupervised multitask learners. OpenAI blog **1**(8), 9 (2019)
13. Varma, V., Shah, R., Kenton, Z., Kramár, J., Kumar, R.: Explaining grokking through circuit efficiency. arXiv preprint arXiv:2309.02390 (2023)

14. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L.u., Polosukhin, I.: Attention is all you need. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc. (2017), `https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`
15. Villegas, R., Babaeizadeh, M., Kindermans, P.J., Moraldo, H., Zhang, H., Saffar, M.T., Castro, S., Kunze, J., Erhan, D.: Phenaki: Variable length video generation from open domain textual descriptions. In: International Conference on Learning Representations (2023)
16. Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R.R., Le, Q.V.: Xlnet: Generalized autoregressive pretraining for language understanding. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 32. Curran Associates, Inc. (2019)