General Comments: We thank the reviewers for their thoughtful and useful comments.

Density of Presentation: Several reviewers commented on the density of the presentation. We will add more high-level explanations particularly to make cascaded decoding more intuitive (R2, R3, R4). We agree with the general spirit of simplicity and note that the main methods in this work are actually quite basic. The pseudocode presented for cascade decoding is nearly identical to the implementation used, and the Markov transformer can be implemented with only a few modifications to any transformer architecture.

Specific Comments:

R1 -Long-Term Dependencies: It is a good point that there is a tradeoff between better inference and longer term dependencies among outputs. We note that this method is a compromise in this regard between NAT and fully autoregressive. It is true that we lack long-term dependencies of very long-term models (GPT-3); however it is an open-question of whether all these dependencies are required for conditional generation. For example: one might imagine introducing latent variables to model global topics, or use cascaded decoding to prune the search space for a fully autoregressive model. We think these are promising directions for future work.

-Length Prediction: Like most NAT methods, our work needs to predict length beforehand, which we can do with linear or non-linear regression. However, unlike most of those works, our length constraint only needs to be a *maximum length*. For MT we predicted this value, but for other settings with more variable length it could be just set to a large constant.

R2 -More powerful CRF modeling: For training speed, we use a locally normalized LM (a special case of a CRF). It is a very interesting suggestion to instead train a globally normalized CRF. This would require running the cascade decoding at training (a difficult but perhaps feasible approach). We think this is interesting future work.

-Relationship to Sun et. al. 2019: We will elaborate on the relationship in our next version. Sun et al introduce a 1st-order CRF on top of a non-autoregressive model. Our work goes beyond a 1st-order CRF and approximates argmax from higher-order CRFs. In addition, we use a single Markov transformer to parameterize all log potentials, instead of using additional side-parameters for pairwise potentials. Lastly, we propose tree decoding to make the parallel complexity sublinear, whereas Sun et al's work is linear.

-Why max marginals and not ngram scores: The problem with using ngram scores is that they do not consider compatibility with other positions. Max-marginal fixes this issue with negligible extra time. On WMT14 En-De val, using ngram scores would get BLEU of 28.42 at 123.48ms, while using max marginals reaches 29.24 at 128.58ms.

R3 -Performance of approach: NAT research methods inevitably sacrifice accuracy for speedups vs fully autoregressive models with serial search. Compared to these methods, we present a new point on the Pareto frontier of accuracy vs speed: many of our results are very close in accuracy while achieving major speedups. Within this space, we believe these numbers are valuable, particularly given that this is a novel and clean approach: for example, while Levenshtein gets a bit better performance on some datasets, it requires hand-crafted policies for training, and additional tricks at inference. On the other hand, our approach requires much less tuning: at training time we simply used the same set of hyper-parameters as training language models, while tuning K and iters only happens at inference (like beam size in beam search).

-How to generate beyond length 3: We run a parallel version of the forward-backward algorithm, each time pruning low-scoring ngram choices at each position, and we gradually increase n from 1 to M. Finally, to generate an entire sentence, we use dynamic programming (Viterbi) to find the sequence with the highest score under the pruned search space. We presented a few examples in appendix A which might be more intuitive.

-Whether $x_{i:j}$ in $\mathcal{X}(x_{i:j})$ should appear at the same position or not: It needs to appear at the same position.

-Tree dynamic programming notations: We will clarify our notations in our next version. C^i_{j,k_1,k_2} is the max possible score of all spans from the $(j * 2^i + 1)$-th position to the $(j + 1) * 2^i$-th position (the length of any span is 2^i), with the constraint of the left end being word k_1 and the right end being word k_2. We compute C^i bottom-up, starting from $i = 0$ and merge adjacent spans in C^i to get C^{i+1} (such that span length becomes $2 * 2^i = 2^{i+1}$). The *prefix* score P^i_{j,k_1,k_2} stores the max possible score of all spans from the *1st* position to the $(j * 2^i + 1)$-th position (also with end constraints), which we compute iteratively top-down using P^{i+1} and C^{i+1}. Conversely, the *suffix* score S^i_{j,k_1,k_2} is the max score of all spans from the $(j + 1) * 2^i$-th position to the *last* position, also computed top-down. Eventually, we use the prefix scores P^0, suffix scores S^0, and log potentials C^0 to calculate max marginals of any edge.

R4 -Performance: Please see comment above for R3. We will also include more recent papers in the next version.

-MT Baselines: We followed the exact settings in fairseq translation examples. The 0.3 BLEU score difference might be due to randomness. Our models use the same machine and settings as baselines, so the comparisons aim to be fair. Our IWSLT14 performance baseline is already much better than existing NAT works.