v2 outline

Methodologically, are we doing as good as we can with classifier guidance today?

Using a gradient means that the update is unnormalized in terms of the rates. We should first learn these two things separately. For the general case of something like edit flows.

But for guidance specifically, we need to do this because it's not tractable to do exact guidance. We can get better by doing exact guidance using order-agnostic sampling. The issue is that this is sequence alphabet size times slower.

So if the goal is to do guidance efficiently, let's start by benchmarking the best possible that we can do using the unconditional model. We'll go down the ladder of step sizes until we get the minimum where we still get decent performance. The goal will be to make something that is much more efficient; it should get more positive hits for the same amount of compute.

This would be a good time to figure out a better way to benchmark these kinds of experiments in silico. Maybe we should try to make a discriminator between ESM's own sequences and the ones from Uniref. We can do this maybe with an ensemble of classifiers built off of sequence models of different origins. Although the weird thing is, such a model should never predict something is not from ESM right? Because either something is fake and it looks like it's from ESM, or it's something that looks natural. It should never be not ESM. Also use folding. And then finally use the clean classifier.

So instead, let's make a classifier that predicts the probability for each mutation. The worry is that this will perform a lot worse than a classifier trained to just predict the logits for one sequence. It's basically doing 20 times the amount of work in the same capacity.

Include a study of going from a regular classifier on sequences to one on the embeddings, like with a different model to with the same model to using the multi-head classifier.

Finally show that framing this as an offline learning problem and training this using the classifier's own direction might be best.

Potentially compare to RL baselines? Maybe compared to LoRA? Should we look at if there is any forgetting in any of these three?

v2

Main strategies: [^15]

- unconditional with aggressive euler
- direct TAG
- use direct exact instead
- normalize TAG and use base-model
- use multihead on base-model
- latter two with tau-leaping

Story risks:

- **Need to check if multihead or normalized TAG will be better**
- Need to understand the related work
- Prob need to check the reweighting baseline for the top-n [^16]
- Don't have experience with guidance on language for safety or code (or anything but the enzyme example)

Classifier guidance is great because as models grow, fewer are open weights. [^1]

summary figures with NFE/sample vs p(y) and p(y)/s of wall clock time

For the broader problem of sampling from a model to maximize some objective function [^2], there are many ways to frame the problem and it may certainly be the case that classifier guidance may not be the best. If we look at some recent results, we may wonder if we are doing it right even. P(y) resulting from our guidance seems unreasonably low sometimes.

insert guidance figures here: dlm coding, enzyme design, stability, promoter design

Normally we write this off as due to models being imperfect, sampling being inexact, or because Bayes' rule only increases P(y) but doesn't maximize it. [^6]

But if our P(y) gains aren't that great, can we just do unconditional sampling and push the time discretization to make each sample cheaper. If rejection sampling suffices we shouldn't bother with guidance right? if the goal is to do guidance efficiently, let's start by benchmarking the best possible that we can do using the unconditional model. We'll go down the ladder of step sizes until we get the minimum where we still get decent performance. The goal will be to make something that is much more efficient; it should get more positive hits for the same amount of compute. [^8]

In this post, I aim to convince you that there is more work to be done and to illustrate a framework with which we can get far better results with guidance without sacrificing on compute efficiency.

Above we saw that classifier guidance doesn't maximize P(Y), but are we doing as good as we can with classifier guidance? Let's check with an in-silico experiment where we use the clean classifier and the generated examples and then we re-weight them to calculate an effective P(Y|X) under the classifier guidance procedure. [^7] Additionally, we'll look at if it converges to some value as we increase the dataset size to get some idea of the variance of this estimate.

Reweighting figure

So what's going on. The standard methodology is to use TAG. We can't use exact classifier guidance because it requires S evals per position, whereas TAG requires only 1 evaluation total. Looking at the distribution of jump sizes in our integration will shed some light. This should be poisson distributed, [^5] and the ones with large number of jumps tend to be the samples that fail. This is *not* just a time discretization issue.

Jump distributions for enzyme and stability, doeesn't change with time-discr

This occurs because the TAG gradients are not normalized like the exact guidance distribution is. [^10] But why is sampling multiple positions problematic in the first place? If we look more closely at those jumps time, each position that transitioned was predicted to increase the probability quite a bit, but together they did't. Why did this happen? There is an important conditional dependency between the positions. When we decode multiple simultaneously, we risk this product of marginals being a poor estimate of the true distribution. [^11]

We can get better by doing exact guidance using order-agnostic sampling.

Show enzyme figure from natbio paper

This is only a bit slower than unconditional sampling, but is now worth the extra computational cost. ~~The issue is that this is sequence alphabet size times slower, so you should actually just do uncondtional sampling and reject poor samples in this case. Even if we use OA-AR sampling [^5] with tau-leaping, we still have S-times more classifier evaluations. (But this may not be a big deal. [^4])~~

Multi-head classifier instead

So instead, let's make a classifier that predicts the probability for each mutation. The worry is that this will perform a lot worse than a classifier trained to just predict the logits for one sequence. It's basically doing 20 times the amount of work in the same capacity. However, as shown earlier, we can build this on top of our denoising model for no additional cost. [^7, ^12]

This should be trained so that you have a classifier for p(y|x) on the average and have p(y|\tilde{x}) per position. [^14]

Now you have something that runs *faster* than TAG (no backwards pass so you can use the whole model embeddings [^13]), is more accurate than the old implementations, and can be sped up with less performance degradation than before.

Finally, we can also layer in tau-leaping [^17] to see how far that gets us.

[^1] Model safety concerns, commercial competitiveness, etc.

[^2] PaVADA? All the MBO stuff are real baselines, including DBaS. Potentially compare to RL baselines? Maybe compare to LoRA as well? Should we look at if there is any forgetting in any of these three? Can always ignore these because it requires access to the weights.

[^3] Should I train/eval everything with the base model dist so that it isn't a point of contention?

[^4] Though maybe this isn't an issue if the classifier can be trained to work well and it's a fraction of the cost of the base model. Then we can do Tau-leaping to get the rest.

[^5] Show proof and show in more detail that the rates of staying don't change, so the rate of transitioning doesn't either. This means you can do fixed batch-size decoding to maximize the use of wall time. This is a beutiful point, right and it also leads you to why TAG fails.

[^6] like we might do with an RL, MBO, or SFT-based method

[^7] there is an additional issue of how good the noisy classifier is (plain small TAG, plain small TAG normalized, plain small exact, plain normalized TAG on base model, multi-head on base model, plain exact on base model; three to train, plain, plain on base, multi-head on base), we can benchmark this by taking model generations, classifying them with the clean classifier and checking the calibration. probably will need to start by looking at the average p(y|x_t) curves to find the times with most change and picking those partial sequences for testing. can actually do that with the existing unconditional and conditional paths

[^8] This would be a good time to figure out a better way to benchmark these kinds of experiments in silico. Maybe we should try to make a discriminator between ESM's own sequences and the ones from Uniref. We can do this maybe with an ensemble of classifiers built off of sequence models of different origins. Although the weird thing is, such a model should never predict something is not from ESM right? Because either something is fake and it looks like it's from ESM, or it's something that looks natural. It should never be not ESM. Also use folding. And then finally use the clean classifier.

[^9] Don't think of classifier as standalone

[^10] What if we just normalize the gradient by taking *e.g.* a softmax

[^11] We should first learn these two things separately. For the general case of something like edit flows.

[^12] Can also try out a path-planning heuristic and offline RL-style training (but these OA+speculative sampling+causal mask and data-guidance+lattice papers should probably be separate)

[^13] Downside of whole model with ESM for TAG is that gradient is more expensive. Whole model TAG is still much better than whole model exact, but can also do exact with multi-head.

[^14] You could def use this constraint to see how well the model has learned these distributions, or you could try to make them consistent by construction/as an auxiliary loss. Or screw it, just learn p(x|y) on top right?? You could also just learn p(y|x') w a softmax and then sum over the p(x'|x) to get p(y|x). Also the original classifier is multi-class right. Can either do same capacity but 20 copies of the output and adjust params accordingly, or do 20 copies of the model and check the computational cost of that. Might be significant.

[^15] Compare with notes in LogSeq and notes on last draft

[^16] For super rare things, you probably need some amount of online RL/MBO/SFT to increase base prob of target class.

[^17] Could you use the gradient of the multi-head as a proxy for path planning?

[^18] Somehow the desired rate should control a scalar multiple on the TAG estimate right? It's like the distance we let ourselves move according to that gradient correlates somehow with the the rate of transitioning away from the masked state in the first place. Should just have that sum of the gradients scaled to be 1.

v1

Guidance can do some amazing things: *show examples from papers*.

- DNA, images, protein stability
- Coding accuracy, harmlessness, avoid bio and security hacks

The base flow model samples an output by learning *transition rates* for each position. These rates tell us how likely the model is to sample a change in the value of that position at the current time. By integrating these transition rates and the resulting state changes, we simulate a CTMC whose terminal distribution at the "denoised time" should approximate a sample from our desired generative distribution.

What classifier guidance does is that it allows us to tilt the distribution of any base model according to an external reward we choose at inference time, *without* training or finetuning the base model. [^4] The way it works is that we reweight the transition rates of the CTMC according to Bayes Rule. We can see this by rewriting the rates in terms of the inferred denoised distribution and a rate term. [^2, ^3]

Equation here

Then the ratio we applied can be used to rewrite the unconditional distribution in the rate in terms of a conditional one.

This basically says that when we find a move (next state to sample) that is better than the average move the base model would make, we should upweight the probability of transitioning to it accordingly. [^5, ^6] When we find such a move, this ratio gets large and if we sample a change at this position at that time, we are very likely to sample the move that is predicted to lead to a high-reward sequence under the guidance model. [^7, ^8, ^12]

The issue is that we are not exactly integrating the CTMC, we do Euler integration. This means we take a fixed Δt and sample the transitions independently per position over the interval, assuming the rates at each position don't change when another position transitions. Not only are the rates not changing for that time interval, but the number of positions that change is now a Poisson random variable. For guidance, this means if I change my sequence in the previous step such that multiple other positions' distributions get very sharp, there is a chance that many of them jump during the same interval and I don't actually get the promised increase in the likelihood of reward. [^9, ^10]

Sampling multiple positions at once doesn't always need to cause issues. In particular, if the reward was linear additive across positions, this would be no problem at all! [^11] In the protein world, we use the concept of epistasis to describe this phenomenon. Epistasis can roughly be thought of as the conditional distributions of two positions being dependent. When doing guidance, by an Occam's razor argument, we would expect greater levels of conditional dependency as the optimization problem gets harder. This means, the more we need guidance, the less it is actually able to help us in the default sampling paradigm.

Before we get to solutions, let's first verify the nature of the problem.

- Toy example of adjacent positions
- Predicted prob of mutation and the actual change when sampled with multiple
- Check that distribution of number of positions changed doesn't change, meaning the total rates aren't either [^13]

There are three strategies we might try, and as alluded to earlier, both have to do with the sampling procedure. The first is to make sure we only sample one position at a time. The easiest way to do this is to use Gillespie sampling, which in this case looks like using my flow-matching model as an OA-AR model. The second, is some form of speculative sampling, where we sample multiple positions but then correct the distribution by thowing away some of the transitions and modifying others afterwards in some cheap-to-compute way. Predictor-corrector sampling could be seen as a particularly simple version of speculative sampling, but its theoretical guarantees are not particularly strong, [^12] making this a good choice for future work. Finally, we can choose the positions to unmask at each time such that they interfere with eachother the least. These path-planning algorithms are still pretty early, but are hard to formally analyze because they inherently modify the flow-matching setup by making it so that we are not interpolating between the noise and denoised distributions by uniformly masking at random. Since the unmasking order is now learned for the data, we don't have an obvious strategy to describe it symbolically.

This post just aims to convince you of the nature of the problem, so we just solve the problem with Gillespie sampling. You can see, even if we use a very small time-step, such that only a modest number of positions change in any given time step, the change in the classifier's probability if other positions are sampled jointly is *far worse* than if those same positions were sampled one at a time.

[^1] Regular guidance on rates is still useful when there is no explicit form in terms of the $p_{1|t}$, but we should still do Gillespie in that case.

[^2] Think you might always be able to write rates as a scaled $r_t(p_{1|t} - \delta)$? Here the rate term just ensures that the necessary transitions to get to X_1 by time 1 occurs.

[^3] Need to review the ICLR paper proof to make sure I know when this holds or not for general flow models.

[^4] Compare to CFG, Masa's works, derivative-free methods, and data-guidance methods. Guidance vs lora, why we don't want to do it with weak classifiers (or when we do--interp), (how this interacts with Haozhe surjectivity--does it have implications for adversarial inputs?), and why we might want to predict the $p(y|\tilde{x})$ as another prediction head per position.

[^5] Connection to Kevin's work

[^6] How do you actually get the numerical issue? If you write this as the conditional distribution, the total outward rate should still be the same. So the issue isn't that multiple things get denoised at once--this should just be a poisson random variable. What might be happening is just when one mutation sharpens the distributions at multiple other positions, each of those transitions could be sampled together even though they are not all good jointly.

[^13] Issue is that you get this even with very small timesteps where you shouldn't sample positions together...

[^7] Note that, if the classifier is doing it's job, it will eventually move the base model to a part of its own distribution from where it can sample uncondiitonally: eventually information about the target class will be baked into the sequence and so the class has nothing to add to the generation process. We can guess when this will be by looking at the percentage of sequence predicted well by the classifier as the proportion of denoised positions increases.

[^8] Also note that the speed at which we sample changes has not been modified since the rate of staying at the current sequence is multiplied by 1.

[^9] Is corrector sampling sufficient to fix this?

[^10] Is the classifier correctly predicting the rewards in the individual case? How calibrated is it again? On the dataset versus generations (where its own label is the determinant)?

[^11] Then there would be a related problem of reward hacking. How does Cassidy's paper on this go again? For proteins we just check if the sequence folds as a measure of realism/"success" in generation.

[^12] Proof?

[^13] We don't necessarily pick the positions that are most important to set somehow to guarantee maximizing $p(y|x)$