Summary: i recommend investigating points 1, 2, 3 below right away. the other points are potentially useful things to try later, if the need arises.

1) I was thinking about why the model doesn't show "dips" at the start of blocks 3 and 4 in the blocked condition (see slide 37). i think this is because the model infers a new latent cause based on the cafe / brewhouse observation, and we have set the model up so that its prediction is *only* based on its MAP estimate of the latent cause, as opposed to having the model make its prediction based on all latent causes, weighted by the probability of each latent cause. sam had recommended the latter but i requested the former. i now think that (as with so many things in life) sam was right and i was wrong, and that we should switch back to the way sam was suggesting (i.e., following equation 5 in sam's document). i think making this change could help with the issue mentioned above (of the model not showing "dips" at the start of blocks 3 and 4) -- after you make the change, the model will make its prediction based on the most likely latent cause but *also* based on the other latent cause (which will make wrong predictions, dragging down performance). i also think this is a relatively "safe" change to make to the model (i.e., it's unlikely to break anything important), because it only affects predictions and not the inference process.

2) i'd like to see what happens to the model predictions on slides 36 and 37 when you keep all of the parameters the same but lower beta_between (varying it smoothly from 100 all the way down to zero). i'm envisioning 5 plots: one each for blocked, interleaved, early, middle, late. for each plot, the y axis should be average prediction accuracy during the final test phase (i.e., averaging across timepoints 160-200) and the x axis should be the value of beta_between. i anticipate that, as you lower beta_between, interleaved test performance is going to get better at some point (as will test performance in the middle and late conditions) and i'm curious what the shape of the curve relating beta_between to test performance will look like. hopefully you will be able to find some values of beta_between that yield intermediate levels of performance in these conditions.

3) in the same spirit as #2, i'd like to see the same kinds of plots where you a) reduce beta_within in a graded fashion, holding the other parameters from slide 36 constant; b) vary alpha in a graded fashion (from 0 upwards, past its current value of 1.4 -- maybe do 0, .5, 1, 1.5, 2, 2.5 … up to 10), holding the other parameters from slide 36 constant

4) this is less of a priority than #1 and #2, but i was thinking more about the problem you identified on slide 34. you were able to resolve it by increasing within-story stickiness. however, i think that another way of addressing this would be to engineer things such that -- instead of the model inferring cause zero at story 43, layer 1 -- it infers a NEW cause (i.e., it creates one latent cause for 0-2-4 and one latent cause for 0-2-3). you should think about what you could tweak to make this happen (e.g., maybe changing alpha would increase the likelihood of the model forking off a new latent cause). to look at this more carefully, i would go back to the parameters that generated the curves shown at the top of slide 34, and look at the probabilities of a) cause 0, b) cause 1, and

c) the new latent cause (cause 2, i guess) for story 43, layer 1. in the example shown on slide 34, cause 0 was the most probable. what would you need to change to make cause 2 the most probable? again, the most obvious thing would probably be alpha, but changing alpha could mess other things up. to be clear, if you end up being able to fit all of the human data well using the high beta_within value used on slide 36, then you don't need to bother with any of this. but if it turns out that using a high beta_within value messes up other things, you may be able to "retreat" to using a lower beta_within value if you can get the model to infer a new latent cause instead of cause 0 at this critical juncture (story 43, layer 1).

5) here is another thing i thought of that might not be necessary, but i nonetheless wanted to mention: currently, as i understand things, you apply beta_between to layer 0 and beta_within to layers 1, 2, and 3. another way of setting things up would be to "store" the z value (i.e., the active latent cause) at the end of each story, and then apply beta_between and beta_within to ALL layers. so, at all layers, when computing the prior, you first determine a) whether the latent cause matches the z value from the *end of the previous story*, and b) whether the latent cause matches the z value from the *previous timestep*. if condition "a" is met then you add beta_between to the prior. if condition "b" is met then you add beta_within to the prior. and if BOTH conditions are met then you add both beta_between and beta_within to the prior. this way of handling beta_between implements a primitive form of memory that spans multiple timesteps within a story (i.e., it allows the model to remember the previous story's latent cause, even if it infers a new latent cause at some point within the current story). i think this could potentially work well with the idea proposed in #1 above -- this new approach to the betas will make the previous latent cause more probable at both layer 0 (which is already happening) but also at layers 1, 2, 3, which should pull down performance at block transitions. the tricky thing is that, if the previous latent cause becomes *too* probable, then the model will just infer that latent cause again, and it will fail to learn in the blocked condition. the goal is to make the previous latent cause *more* likely (so it messes up performance at block transitions, given the modification described in #1 above) without making the previous latent cause the *most* likely latent cause (i.e., to ensure that the model learns, the most likely latent cause either needs to be cause 1 or cause 2).

6) i just wanted to mention (for potential future use) that my idea of adding a "learning rate" might be useful for fine-tuning fit to human data. this would involve incrementing the M_{tkij} tables by *lrate* instead of by 1 when a transition is taken. put another way, the current way of doing things is equivalent to an *lrate* of 1, but you could also use a smaller *lrate* if you wish. based on equation 4 in sam's document, i'm confident that these lrate changes won't simply cancel out (because the M tables are not the only thing in the equation -- there's also the lambdas…). i'm not sure about whether this change will also require multiplying N_{tk} by *lrate* in equation 1 in order to keep the math "correct".

7) let's say that, for the analyses shown in 3 and 4, you find that changing parameters tends to affect performance in a very "high gain" way (e.g., as you vary beta_between,

maybe interleaved test performance will jump right from .50 to 1.00 without spending much time in an intermediate range). if the model behaves like this, it might be hard to fit human data where human performance is in an intermediate range (like .70). one simple way to address this problem would be to say that each parameter value is sampled (for a given simulated participant) from a gaussian distribution around a mean -- if the gaussian is wide enough (encompassing parameter vals that give rise to .50 accuracy and also vals that give rise to 1.00 accuracy) then -- averaging across simulated participants -- you will be able to get intermediate levels of accuracy.