

## 486 A Argument-Function Processing in Other Models

487 In Section 3 we show that GPT2-Medium and Bloom promote the in-context ‘argument’ token to  
 488 some function before promoting the answer to that function. In figure 8 we show that this effect  
 489 is present across other models as well in the three tasks we test. Qualitatively, we find that the  
 490 pattern is more prominent in models that have more layers, likely because we are able to get more  
 491 measurements after the FFN updates, so it is less likely that entire argument formation stage happens  
 492 within a single layer (i.e., after the attention module update – we only take measurements after the  
 493 FFN update for simplicity). In the extractive task setting, we would not expect the model to go  
 494 through argument-function processing in order to reach the prediction, since it already appears in  
 495 context (although this does not preclude it from doing so – it is still a valid way to retrieve the required  
 496 information). We see that this X shaped pattern disappears when we plot the argument-answer curves  
 497 for the extractive world capitals data, as shown next to the abstractive setting in Figure 9.

498 We repeat the random tokens task on GPT-J using the same stimuli as in the main paper to select  $\vec{o}$   
 499 vectors. We find that we can locate  $\vec{o}$  vectors occurring in other models, however the success rate  
 500 varies for the tasks that we evaluate in this work. Results are shown in Figure 10. Although the  
 501 uppercasing function works very well, we get weaker responses for the past tense and world capitals  
 502 mappings. One explanation could be that these tasks are not solved with an as-general solution as in  
 503 GPT2, but the process for carrying out this intervention depends on hyperparameters which are often  
 504 model-specific (i.e., the exact layer at which to perform the intervention), so future work is needed to  
 505 understand where differences between these models lie.

## 506 B Additional Results on Ablating FFNs

507 We include the results for all six models we test for the FFN ablation study for both the colored  
 508 objects task (Figure 11) and the world capitals task (Figure 12). We find that the trend of abstractive  
 509 performance dropping off far before extractive performance is reflected across all models.

### 510 B.1 $\pm o_{case}$ Intervention on Colors

511 As illustrated in the example in Figure ??, adding  $o_{case}$  to the residual stream ( $x_{19} + o_{case}$ ) has the  
 512 effect of capitalizing the first letter in the word ‘brown’. Similar to the results in Sections 2.2 and 2.2,  
 513 we find that adding  $o_{case}$  to the residual stream has the effect of uppercasing the token prediction  
 514 on arbitrary contextualized representations in the mid layers of GPT2-Medium. However, we also  
 515 find that lowercasing the first letter can be accomplished by *subtracting* it. Qualitatively, this works  
 516 much the same way as adding the  $\vec{o}$  vectors previously discussed. We show this effect empirically, by  
 517 showing the difference between replacing the FFN updates in GPT2-Medium with either positive or  
 518 negative  $o_{case}$  (having the effect of adding or subtracting from the residual stream).

519 We progressively remove FFNs from the top of the model, and show the effect of adding or subtracting  
 520  $o_{case}$  in Figure 13. In the abstractive case, we find that accuracy is greatly boosted when adding  
 521  $o_{case}$  which we identify as implementing an uppercasing function, and reflects the results in Sections  
 522 2.2 and 2.2. We find that we can replace the top third of GPT2-Medium FFN layers (FFNs in layers  
 523 16-24, around 20% of all parameters) with  $+o_{case}$  to gain 25% in total accuracy (from 4.5% to 29.5%)  
 524 and recovering to 72% of the performance of the un-ablated model (41%). Conversely, if we subtract  
 525  $o_{case}$  in the abstractive setting to encourage lowercasing (i.e., encouraging the model to output a  
 526 lowercased answer when the answer it should have a capital first letter), the model immediately hits  
 527 0% performance. We see the opposite effect in the extractive setting, where adding  $o_{case}$  hurts  
 528 performance to a greater degree than subtracting it. According to our results presented so far, we  
 529 would expect FFNs to be unnecessary for solving the extractive dataset examples, which is possibly  
 530 why performance is degraded in both cases we intervene, but we don’t test this idea in this work.

## 531 C Effect on Zero-shot Performance

532 We find that intervening on the model with  $\vec{o}$  vectors has applications in controllable generation, that  
 533 is, guiding the generation process towards some relevant text. We showed this was the case in Section  
 534 4, but we can also apply this idea to the context of zero-shot learning. When we provide in-context  
 535 examples, we are also providing the output format of the prompt. Consider the example “Q: What

## Argument-Function Processing in the Last Token across Task/Models

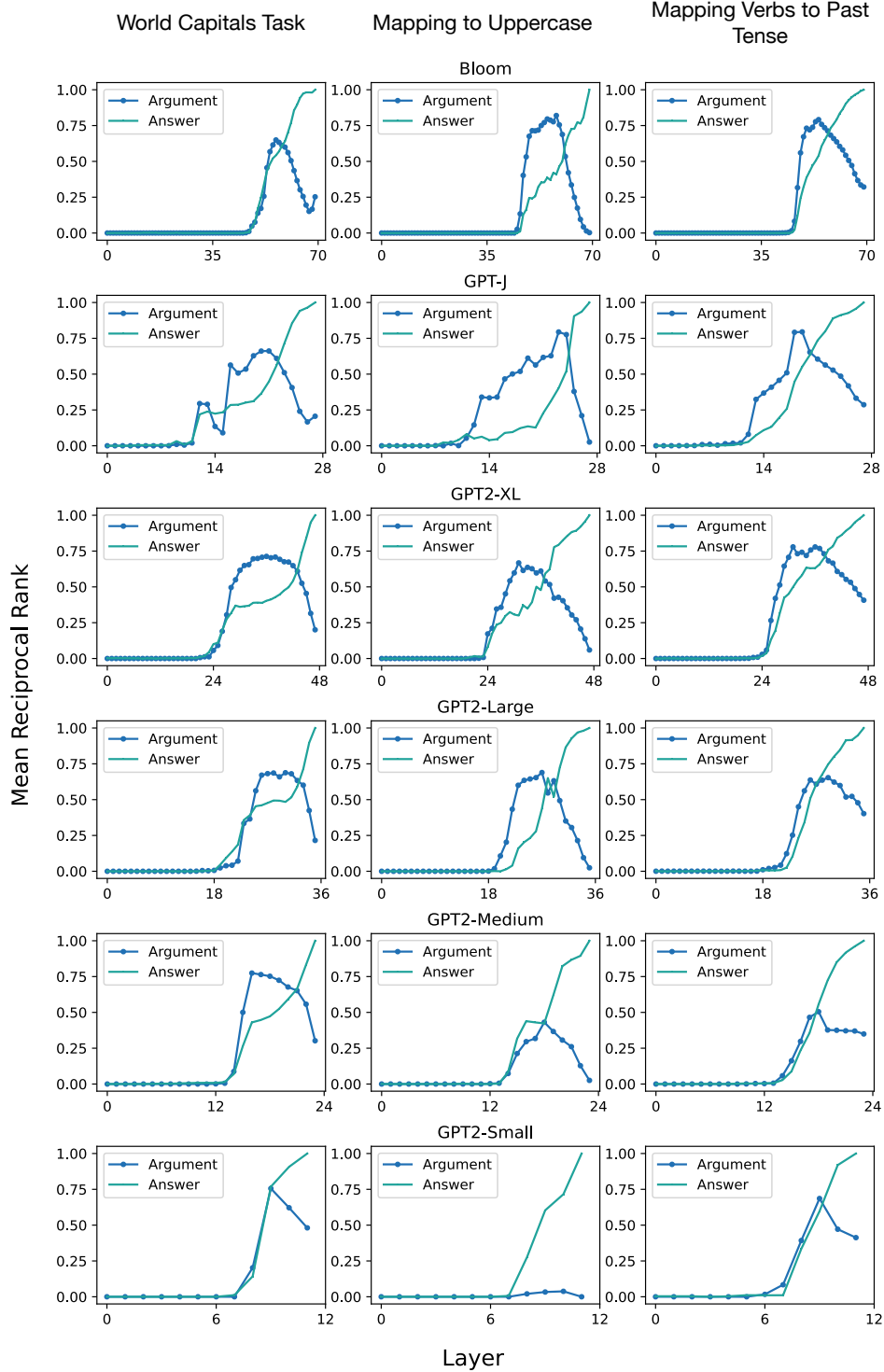


Figure 8: Across several model architectures and tasks, we find evidence that on average, the argument (which appears in context) rises to the top of the vocab distribution before crossing with the answer to the task. We describe this as argument-function processing where the argument to some function is represented in the residual stream before some update from the model is added to it to produce the output of that function. Qualitatively, we observe that models with more layers display this pattern more prominently.

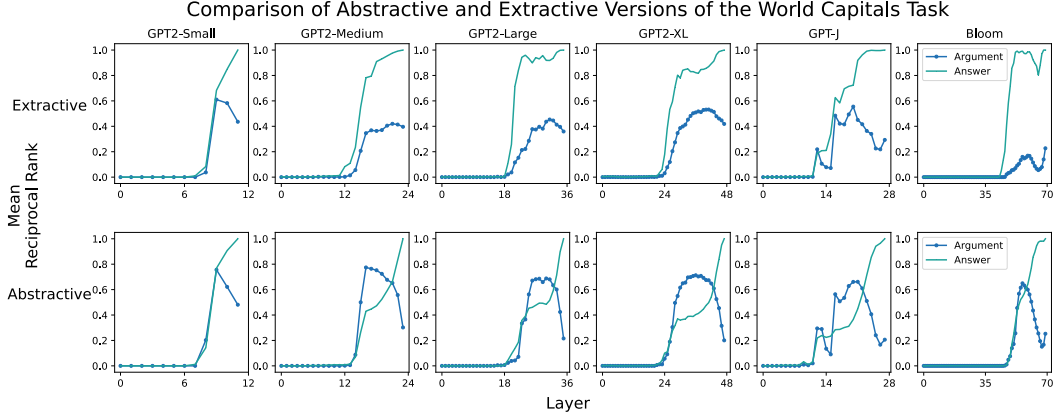


Figure 9: The ‘X’ pattern of argument and answer tokens crossing in the course of the forward pass is the characteristic pattern in argument-function processing. In the main text, we show how the models we test use this type of processing to recall the capital cities of locations. When we make the task extractive (by including the correct capital in the given context), the model does not have to setup an argument and function in order to get the answer, and the pattern disappears. This highlights the differences we describe in processing extractive and abstractive tasks. Both datasets are filtered for examples where the models were correct.

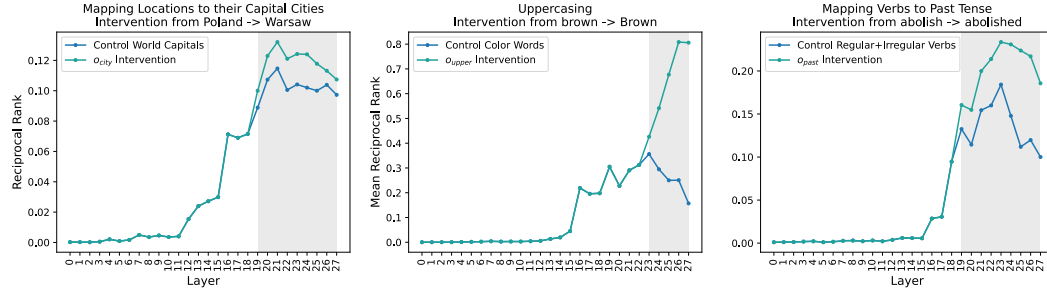


Figure 10: We use the same stimuli to extract  $\vec{o}$  vectors on GPT-J. Results are similar for the uppcasing function, but only very weakly positive on the world capitals task.

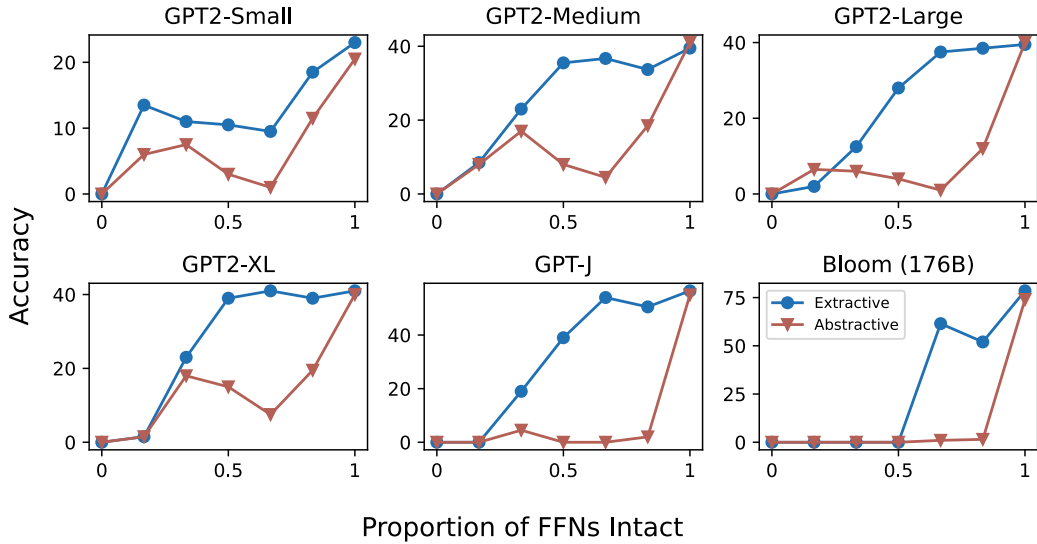


Figure 11: Results of removing FFN sublayers for the colored objects task for all models.

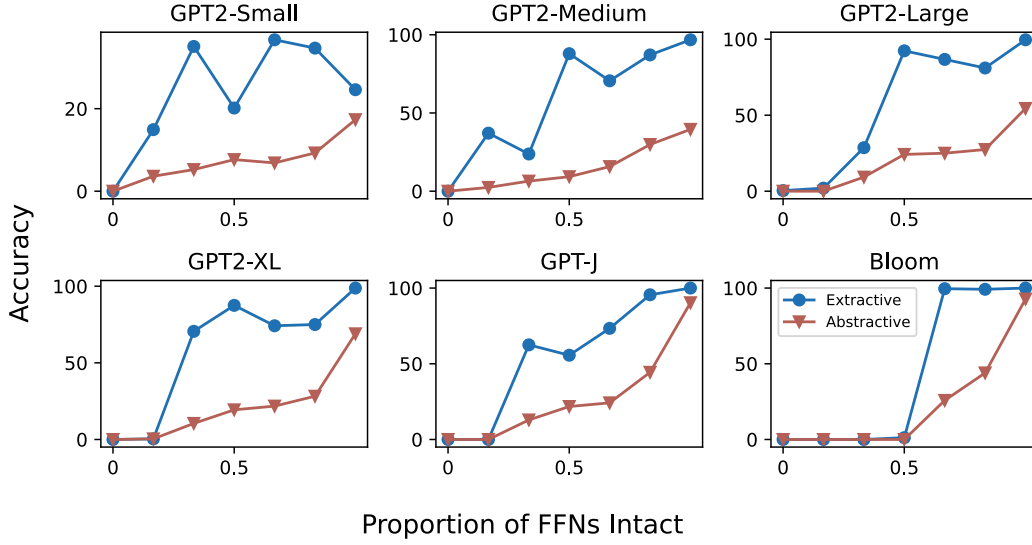


Figure 12: Results of removing FFN sublayers for the world capitals task for all models.

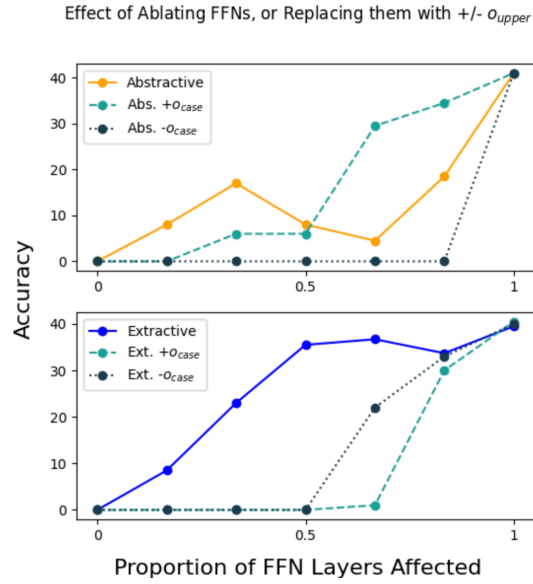


Figure 13: Replacing FFN updates with  $+o_{case}$  helps recover accuracy in abstractive tasks where the answer is expected to be uppercase compared to subtracting it or ablating the FFNs. In extractive tasks, the task is primarily solved by attention modules and adding or subtracting  $o_{case}$  only hurts performance.

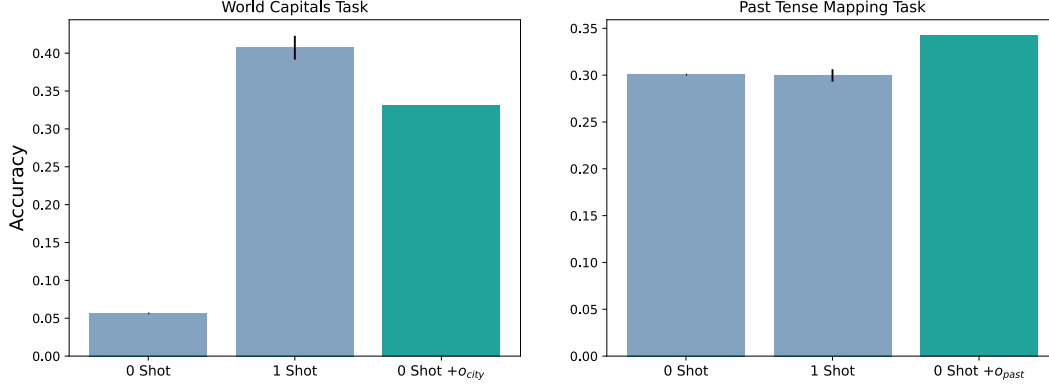


Figure 14: By replacing FFN networks with the corresponding  $\vec{o}$  vectors, we show that we can improve zero-shot performance by taking advantage of the model going through argument formation in the zero-shot setting.

is the capital of Poland? A:". unlike the one shot example given in Figure 2, there is no indication that the next word should be "Warsaw" over continuing the generation as a complete sentence "The capital of Poland is Warsaw", which is what GPT2-Medium actually generates. If we decode at every layer, as is shown in Table C we can see that the model still goes through argument formation despite preferring to generate the full sentence. We can take advantage of this behavior by replacing the FFN layers in the later layers with  $o_{city}$  in order to guide the generation to the expected response of immediately generating the capital. We can perform this experiment on the past tensing task as well.

Layer	Top Token
0	(
1	A
2	A
3	A
4	A
5	A
6	A
7	A
8	A
9	The
10	The
11	The
12	The
13	The
14	The
15	The
16	The
17	The
18	Poland
19	Poland
20	Poland
21	Poland
22	Poland
23	The

Table 1: These are the top tokens per layer in GPT2-Medium on the example zero-shot Poland example

Results on the zero-shot tasks are shown in Figure 14. We find that on the world capitals task, we can greatly improve the propensity of the model to output the expected answer by performing an  $\vec{o}$  vector intervention, improving zero-shot performance from 5.6% to 33.0%. On the past tense mapping task, where perhaps the output format is more obvious from the prompt, the zero and one shot performances are about equal, but we still see a modest improvement over the one shot results of about 4.2%. Although the tasks are very simple, we achieve this by effectively ablating FFN layers (layers 19-23) and precomputing their activations, suggesting it might be possible to edit models extensively to limit their expressiveness to one type of output while also making them more efficient. We are optimistic about future work in this area.

## D Effect of Layer Choice on Intervention Results

In the main text, we replace FFNs starting at either layer 18 or 19 GPT2-Medium to the end (indexed at 0). We find that intervening on only one layer promotes the output token, but not to the top of the distribution. One possibility is that the model makes gradual updates that are pushing the token representation in generally the same direction (Jastrzebski et al., 2017). In Figure 15, we show that adding any of the  $\vec{o}$  vector interventions at any single layer at 18 or afterwards, there is a roughly equivalent increase to the average reciprocal rank of the target word.

## E Effect of Tokenization on the Effectiveness of $\vec{o}$ Vectors

The tokenizer can split one word into multiple subtokens, such as "Purple" into the tokens "Pur" and "ple". This occurs with words that were less frequent in the training data. We find that this process

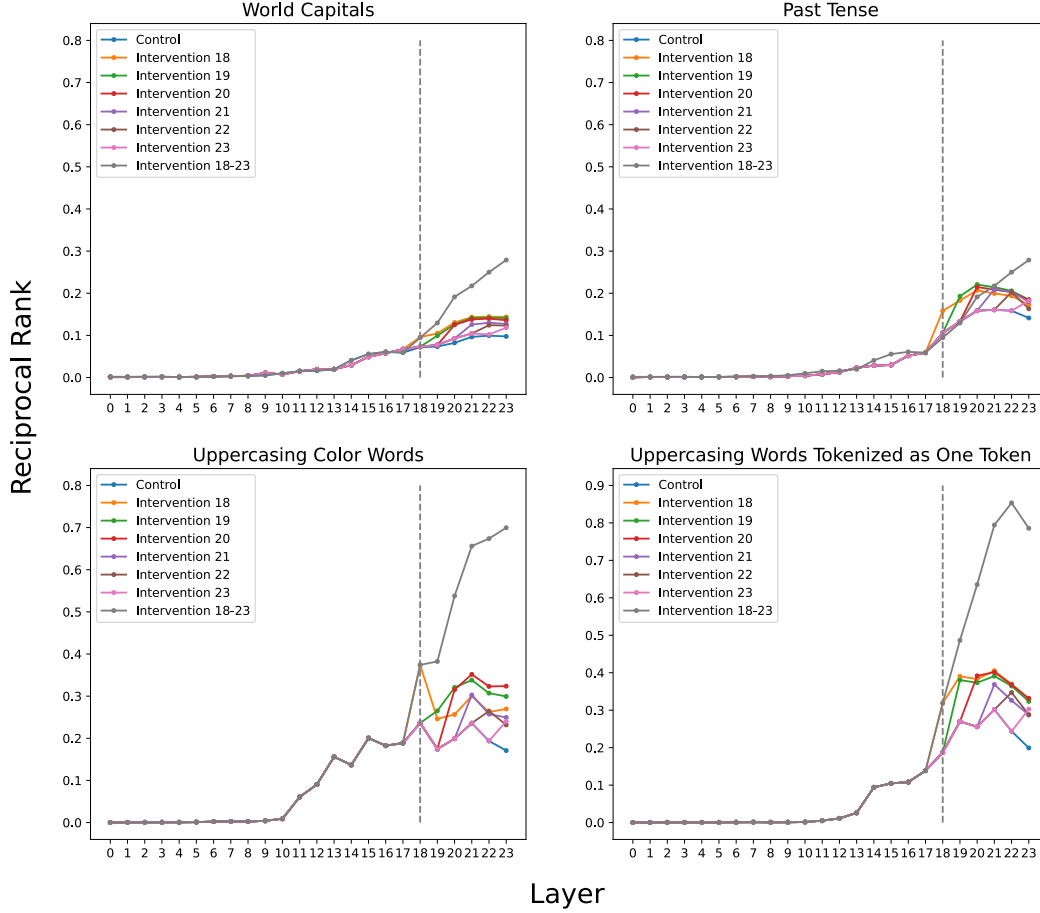


Figure 15: Replacing any individual FFN update is worse than replacing all of them. This supports the idea that networks made gradual updates to their representations, and that the  $\vec{o}$  vectors we extract behave this way as well: multiple similar updates are made  $k$  layers in a row. Interestingly, the average boost to the reciprocal rank is about the same regardless of which single layer we apply the update at, suggesting that this range of FFNs are operating in same space.

has a generally negative effect on the performance of the intervention we perform. Intuitively, if we are trying to use  $\vec{o}_{upper}$  to capitalize the “purple” token into “Purple”, it must map from “purple” (one token) to “Pur”. It seems less obvious, then, that the embeddings would encode a linear relationship between these two, since “Pur” is a subtoken in many other words. We explore this specific phenomenon on the random tokens task from Section 4 with the  $\vec{o}_{upper}$  intervention. We take 100 single token words that capitalize to a single token, and 100 others that capitalize to words that break down into multiple tokens. Our results can be seen in Figure 16. We find that tokens that get broken up into multiple tokens are less probable than for tokens that capitalize to single token forms.

## F Compute

All models were run on NVidia RTX 3090s; Bloom was run locally on 3090s in float16 with CPU offloading.

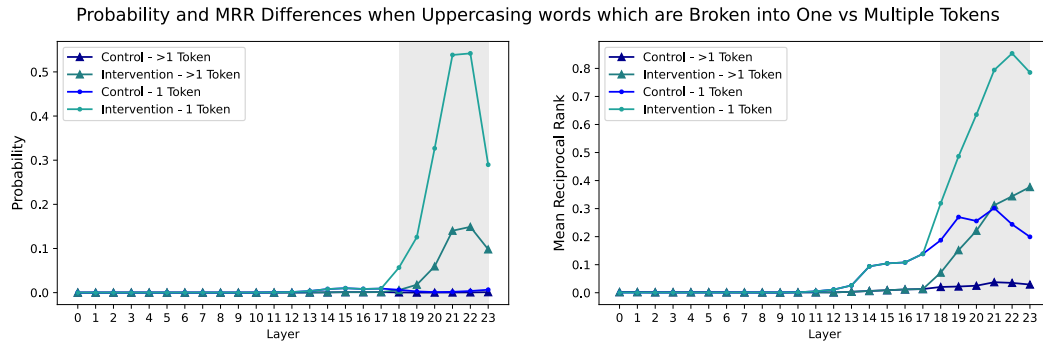


Figure 16: When the uppercase version of a word gets broken down into multiple subtokens, mapping to that token becomes much less probable and is generally harder of an association for the model to make.