Supplemental Material

A Adversarial Filtering Setup

In this subsection, we provide some more details regarding the Adversarial Filtering experiments. Our version of Adversarial Filtering is mostly the same as Zellers et al. (2018). Details:

a. On each iteration, we split the dataset up into 80% training and 20% testing. We don’t do anything special for this split (like looking at the video/article IDs).

b. For ActivityNet, we use \( k = 9 \) assigned indices for every example. (This corresponds to the number of red columns in Figure 2). For WikiHow, we used \( k = 5 \), since we found that there were fewer good endings produced by the generators after scaling up the sequence length.

c. Similarly to Zellers et al. (2018), we train the AF models in a multi-way fashion. Since we use BERT-Large as the discriminator, this matches Devlin et al. (2018)’s model for SWAG: on each training example, the model is given exactly one positive ending and several negative endings, and the model computes probability distribution over the endings through a softmax. However, we also wanted to always report 4-way probability for simplicity. To do this, we train in a 4-way setting (the training set is constructed by subsampling 3 wrong answers from the set of \( k \) that are currently assigned to each example). The accuracy values that are reported are done so using the first 3 assigned negatives in dataset \( D_{test} \).

d. Sometimes, BERT never converges (accuracy around 25%), so when this happens, we don’t do the reassignment.

B GPT Setup

We generate our dataset examples from OpenAI GPT. We finetune the model for two epochs on WikiHow, and 5 epochs on ActivityNet, using the default learning rate of (Radford et al., 2018). Importantly, we generate randomly according to the language model distribution, rather than performing beam search – this would bias the generations towards common words. For the WikiHow endings, we used Nucleus Sampling with \( p = 0.98 \), which means that the probability weights for the tail (those tokens with cumulative probability mass < 0.02) are zeroed out (Holtzman et al., 2019).

C BERT setup

We extensively study BERT in this paper, and make no changes to the underlying architecture or pretraining. For all of the experiments where we provide context, we set up the input to the BERT model like this:

[CLS] A woman is outside with a bucket and a dog. The dog is running around trying to avoid a bath. [SEP] She gets the dog wet, then it runs away again [SEP]

In the case where only the ending is provided, we adopt the BERT-style ‘single-span’ setting: [CLS] She gets the dog wet, then it runs away again [SEP]

D A discussion on BERT

Hyperparameters and Instability

It is worth noting that many of our experiments some instability. On the SWAG experiments, we use the same hyperparameters as (Devlin et al., 2018) - these generally work very well. However, we find that they become a bit unstable when crossing over to make HellaSwag. Here, we discuss some strategies and insight that we picked up on.

a. We use a batch size of 64 examples rather than 16, and warm the model up for 20% of the dataset (rather than 10%). This helps the model adapt to SWAG more gradually, without diverging early on.

b. For the Adversarial Filtering experiments (for both WikiHow and ActivityNet), we randomize some of the hyperparameters on each iteration. We sample a learning rate between \( 1e^{-5} \) and \( 4e^{-5} \), using a log-uniform distribution. These outer ranges were recommended from the original BERT paper. Additionally, with probability 0.5 we use the cased model (where the input isn’t originally lowercased before tokenization), rather than the uncased model.

c. During adversarial filtering, we used 3 epochs. However, we found that adding more epochs

13The only exception is for the plots where we vary the number of training examples. In this case, we don’t want to disadvantage the trials without much training data (since this would allow for fewer parameter updates). To remedy this, we continue training for 10 epochs and report the best validation performance over the entire training history.
helped the model during fine-tuning on the final dataset HellaSwag. Our best configuration uses 10 epochs.

d. While fine-tuning on HellaSwag we used a learning rate of $2 \times 10^{-5}$.

E Human validation

We performed human validation using the same setup as (Zellers et al., 2018). Humans get six answers to choose from, of which exactly one is the true ending and the other five are from AF. We found that multiple rounds of human validation were especially helpful on ActivityNet. However, it helps to do the human validation in an intelligent way: if the first worker is confused, the answer should be replaced before it goes to the next worker. This is a hard problem, so we adopt the following approach:

a. We use best practices on mechanical turk, paying workers fairly (up to 37 cents per HIT on WikiHow). We also used a qualification HIT that was autograded to help filter for workers who are good at the task. Workers who tended to prefer the generated endings over the real ones were dequalified from participating.

b. For each worker, we use the summary of their performance so far to estimate $P(\text{answer } i \text{ is right} | \text{worker rates } i \text{ as best})$. We can then use this to estimate how confident we are in each answer choice: we want to be confident that workers will not prefer the wrong answers. Also, this allows us to aggregate performance across crowd workers, by multiplying the probabilities for each answer choice.

c. On each round of filtering, we keep the 3 wrong endings that workers least prefer (based on the probability scores, along with the right ending. The other two endings are new ones.

Particularly on ActivityNet, we found that there are some contexts where the ground truth answer isn’t liked by workers. To fix this, we end up taking the best 25k examples from ActivityNet and the best 45k from WikiHow. (By best, we mean the ones with the highest probability that workers will predict the true answer, versus the three easiest-to-guess negatives, as judged by the Naive Bayes model). We make Figure 7 (‘The road to HellaSwag’) by doing this process (taking the best examples) for each dataset, while varying the number of annotators that are used for getting the scores for each ending. (In the case where there are 0 annotators, we get a random sample).

F Human Evaluation

We do a human evaluation while giving workers the exact same task as is given to the models. Workers are given five endings, and must pick the best one. We obtain human evaluation numbers by combining 5 turkers together, with a majority vote.

We found that the biggest differences in difficulty in humans were due to domain (WikiHow is easier than ActivityNet). To account for this, we did the human evaluation over 200 examples from WikiHow, and 200 examples from ActivityNet, for each number of previous validators as shown in Figure 7 (0, 1, or 2). To report the accuracy of a split that’s mixed between WikiHow and ActivityNet, we use the following formula:

$$\frac{\text{acc}_{\text{WikiHow}} \cdot N_{\text{WikiHow}} + \text{acc}_{\text{ActivityNet}} \cdot N_{\text{ActivityNet}}}{N_{\text{WikiHow}} + N_{\text{ActivityNet}}}$$

Here, acc refers to the accuracy on each dataset as judged by humans, and $N$ is the number of examples from that dataset in the split.

G More examples

We additionally have more validation examples, shown in Figure 2.

H In-Domain and Zero-Shot categories

See Figure 13 for a closer look at the dataset categories.
Table 2: Example questions answered by BERT-Large. Correct model predictions are in **blue**, incorrect model predictions are *red*. The right answers are **bolded**.
Figure 13: Examples on the in-domain validation set of HellaSwag, grouped by category label. Our evaluation setup equally weights performance on categories seen during training as well as out-of-domain.